11-02-00

EXPRESS MAILING LABEL NO: EL327514237US

JC957 U.S. PTO 11/01/00

Date: **November 1, 2000**
Atty Docket No: **102689-61/00-U0082**

BOX PATENT APPLICATION
Assistant Commissioner for Patents
Washington, D.C. 20231

Sir:

JC914 U.S. PTO 09/703856 11/01/00

Transmitted herewith for filing is the **patent application** of:

Inventor(s): **DARRYL BLACK AND THOMAS R. PERRY**
Entitled: **ACCESSING NETWORK DEVICE DATA THROUGH USER PROFILES**

Enclosed are:

☒ Patent application (including **247** sheets of Specification; **8** sheets of Claims (Claims **#1 - 39**); **1** sheet of Abstract; and **212** sheets of ☒ informal ☐ formal drawings (one set) **(FIGS 1, 2a-2b, 3a-3m, 4a-4z, 5a-5z, 6a-6p, 7a-7y, 8a-8e, 9a-9n, 10a-10i, 11a-11x, 12a-12c, 13a-13d, 14a-14f, 15, 16a-16d, 17-30, 31a-31c, 32a-32c, 33a-33d, 34a-34b, 35-40, 41a-41c, 42-59, 60a-60n, 61a-61b, 62)**;

☒ Title Sheet for Patent Application;

☒ Declaration and Power of Attorney (☒ **executed** ☐ **unexecuted**);

☒ Verified Statement Claiming Small Entity Status;

☐ Information Disclosure Statement (w/Form PTO-1449 and copies of referenced documents);

☒ Assignment (w/Recordation Cover Sheet) of the invention to **Équipe Communications Corporation**;

☒ Check in the amount of **$40.00** representing Assignment Recordal Fee;

☒ Check in the amount of **$526.00** representing filing fee based on

☐ **large entity status** ☒ **small entity status**;

☒ Bibliographic Data Sheet; and

☒ Postcard receipt.

CLAIMS FILED:	MINUS BASE:	EQUALS EXTRA CLAIMS:	RATE:	BASIC FEE $ 710.00
Total: 39	20	**19**	x $ 18.00	**$ 342.00**
Independent: 3	3	**0**	x $ 80.00	**$ 0.00**
Multiple Dependent Claims (1st presentation)			$270.00	**$ 0.00**
SUBTOTAL FILING FEE				**$ 1052.00**
Small Entity filing, divide by ½ (Note: Verified Statement must be attached)				**$ 526.00**
TOTAL FILING FEE				**$ 526.00**

☒ The Commissioner is hereby authorized to charge payment of any additional filing fees under §1.16 associated with this communication or credit any overpayment to Deposit Account No. **141449.**

SUBMIT IN DUPLICATE

NUTTER, McCLENNEN & FISH, LLP
One International Place
Boston, Massachusetts 02110-2699
Telephone: **(617)439-2514**
Facsimile: **(617)310-9514**

Attorney/Agent of Record: **Reza Mollaaghababa**
Registration No: **43,810**

018657.1

Express Mail Label # EL327514237US

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICANT:	Darryl Black, et al.	**ATTORNEY DOCKET NO:** 102689-61/00-U0082
APPLICATION NO:	Pending	**EXAMINER:** Not Yet Assigned
FILED:	Herewith	**GROUP NO:** Not Yet Assigned

ENTITLED: ACCESSING NETWORK DEVICE DATA THROUGH USER PROFILES

Assistant Commissioner for Patents
Washington, D.C. 20231

STATEMENT CLAIMING SMALL ENTITY STATUS

Sir:

THE UNDERSIGNED STATES:

Exclusive rights in the above-identified invention reside in the "small entity(ies)" defined and named below, and "small entity" fees are appropriate. Qualification as a small entity is based upon the statement(s) below:

SMALL BUSINESS CONCERN

The below-identified small business concern qualifies as a small business as defined in 13 C.F.R. § 121.3-18, and reproduced in 37 C.F.R. § 1.9(d), for purposes of paying reduced fees under Sections 41(a) and (b) of Title 35, in that the number of employees, including those of its affiliates, which does not exceed 500 persons, and it has not assigned, granted, conveyed or licensed, and is under no obligation under contract or law to assign, grant, convey or license, any rights in the invention to any person who could not be classified as an independent inventor if that person had made the invention, or to any concern which would not qualify as a small business concern under 37 C.F.R. § 1.9(d) or a nonprofit organization under 37 C.F.R. § 1.9(e).

Concerns are affiliates of each other when, either directly or indirectly, one concern controls or has the power to control the other, or a third party controls or has the power to control both. The number of employees of the business concern is the average over the fiscal year of the persons employed during each of the pay periods of the fiscal year. Employees are those persons employed on a full-time, part-time or temporary basis during the previous fiscal year of the concern.

The undersigned acknowledge(s) the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate (37 C.F.R. § 11.28(b)).

The below-signing individual(s) hereby declare(s) that (he, she, they) are authorized to execute this statement on behalf of the small entity; that all statements made herein of (his, her, their) own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application, any patent issue thereon, or any patent to which this verified statement is directed.

Sheet 2 of 2

Express Mail Label # EL327514237US

Name of Small Entity:

Équipe Communications Corporation

Address of Small Entity:

One Hundred Nagog Park, Acton, MA 01720

Name of Person Signing:

Patricia Davis, Esq.

Title of Person Signing:

Corporate and IP Counsel

Signature: (Please sign and date in permanent ink.) **Date signed:**

x [signature] x 10/26/00

917826.1

APPLICATION

FOR

UNITED STATES LETTERS PATENT

Entitled:

ACCESSING NETWORK DEVICE DATA THROUGH USER PROFILES

Inventors:

Darryl Black

and

Thomas R. Perry

NUTTER, McCLENNEN & FISH, LLP
One International Place
Boston, MA 02110-2699
Telephone (617) 439-2000
Facsimile (617) 973-9748

Express Mail Label No.: EL327514237US

EXPRESS MAIL NO. EL327514237US
Docket No. 102689-61/00-U0082
Date of Mailing: November 1, 2000

ACCESSING NETWORK DEVICE DATA THROUGH USER PROFILES

This application is a continuation-in-part of U.S. Serial Number _____, filed October 11, 2000, entitled "Utilizing Managed Object Proxies in Network Management Systems", still pending.

Background

To configure or manage a network device in a telecommunications network, a user typically connects directly to the network device through a network management system. Most network management software is based on the Simple Network Management Protocol (SNMP), and a user connects to a network device by issuing an SNMP command, for example, read/write or read only, followed by a password in the form of a community string. Often, only one community string is used to access the network device, and thus, all users allowed to access the network device must "share" the community string. Sharing the community string and directly connecting a user to the network device reduces the security of the network device.

More advanced systems allow multiple SNMP community strings to be used to gain access to a network device. Generally, these community strings are stored in a table within the network device. Where a large number of users may be connected to the device, however, community strings may still need to be shared or the table in the network device must be very large.

Although one or more community strings may be shared by multiple users and used to access the same network device, community strings are normally different for each network device. Thus, to connect to multiple network devices, a user must separately connect (i.e., issue an SNMP command and community string) to each network device. This may take considerable time and requires the user to remember multiple community strings.

In addition, if the one or more community strings needed to access a network device must be changed – for example, in response to a security problem – every user allowed to access the device must be notified of the community string change. If the community string is changed quickly, then some users may be denied access until they receive the notification including the new password. If the community string is not changed until after an amount of time necessary to notify all users, then until the change is implemented, the security problem will continue.

Summary

The present invention provides a method and apparatus for accessing network device data through user profiles. User profiles may be created by network administrators, and the corresponding user profile data may be stored in a central network management system (NMS) database. When a user requests data from a particular network device, the NMS utilizes the user profile data in the central database to access the network device and retrieve the required network device data. Since the user profile data is stored in a central database, the user may log into the NMS from any location and connect to any network device in the network. Further, the user profile data may be used to limit which network devices and which network device configured resources the user may access as well as the user's access level.

In one aspect, the present invention provides a method of managing a telecommunications network including storing user profile data corresponding to a user profile in a first data repository, storing network device data corresponding to a network device in the telecommunications network in a second data repository, detecting a request from a user for network device data corresponding to the network device, where the user request is associated with the user profile, generating a data access request to the second data repository utilizing the user profile data from the first data repository and retrieving network device data from the second data repository in accordance with the user request. The first data repository may be a central data repository, the second data repository may be embedded within the network device and both the first and second data repositories may be databases. The method may also include displaying the retrieved network device data in a user interface. The user profile data may include an IP address or DNS name assigned to the network device, and may also include a port identification for a port on the

network device. The user profile data may include a group access level and may also include a password corresponding to the group access level for gaining access to the network device, and the user profile data may include a simple network management protocol (SNMP) community string.

The user profile data may include one or more group names corresponding to the network device and generating a data access request to the second data repository using user profile data from the first data repository may include generating a data access request to the second data repository for each group name in the user profile data and retrieving network device data from the second data repository in accordance with the user request may include, for each data access request, searching the second data repository for the group name corresponding to the data access request and retrieving network device data from the second data repository corresponding to the group name if the group name corresponding to the data access request is found in the second data repository. The network device data retrieved from the second data repository may be configured resource data. Retrieving network device data from the second data repository corresponding to the group name may include using the group name to dynamically determine which data in the second data repository is retrieved. The first and second data repositories may be databases and retrieving network device data from the second database corresponding to the group name may include using the group name in a database query to actively filter which data in the second database is retrieved, and generating a data access request to the second database for each group name in the user profile data from the first database may include generating a where clause including the group name and sending the where clause to the second database. The first and second data repositories may be relational databases and the user profile data may be stored in at least one table within the first database and network device data may be stored in at least one table within the second database.

The method may further include generating a user profile logical managed object (LMO) including at least a portion of the user profile data from the first data repository and generating a data access request to the second data repository utilizing the user profile data from the first data repository may include generating a data access request to the second data repository utilizing the user profile data from the user profile LMO. Prior to generating a user profile LMO, the method may

further include detecting a user log-on request. Detecting a request from a user for network device data may include detecting the user request through a network management system (NMS) client and sending the user request from the NMS client to an NMS server, where the NMS server generates the data access request to the second data repository utilizing the user profile data from the first data repository and retrieves network device data from the second data repository in accordance with the user request, and the method may further include sending the retrieved network device data from the NMS server to the NMS client. The method may further include generating a user profile logical managed object (LMO) at the NMS server, where the user profile LMO includes at least a portion of the user profile data from the first data repository, sending the NMS client the user profile LMO and, where generating a data access request to the second data repository utilizing the user profile data from the first data repository may include generating a data access request to the second data repository utilizing the user profile data from the user profile LMO. The method may instead include generating a user profile LMO at the NMS server, where the user profile LMO includes at least a portion of the user profile data from the first data repository, generating a client user profile LMO at the NMS server, where the client user profile LMO includes at least a portion of the user profile data from the first data repository in a format expected by the NMS client, sending the client user profile LMO to the NMS client and where generating a data access request to the second data repository utilizing the user profile data from the first data repository may include generating a data access request to the second data repository utilizing the user profile data from the client user profile LMO.

The user request may be a first user request, the network device may be a first network device and the data access request may be a first data access request, and the method may further include storing network device data corresponding to a second network device in the telecommunications network in a third data repository, detecting a second user request from the user for network device data corresponding to the second network device, wherein the second user request is associated with the user profile, generating a second data access request to the third data repository utilizing the user profile data from the first data repository and retrieving network device data from the third data repository in accordance with the second user request. The user profile data may be first user profile data, the user may be a first user, the user request may be a first user request and the data access request may be a first data

access request, and the method may further include storing second user profile data corresponding to a second user profile in the first data repository, detecting a second request from a second user for network device data corresponding to the network device, where the second user request is associated with the second user profile, generating a second data access request to the second data repository utilizing the second user profile data from the first data repository and retrieving network device data from the second data repository in accordance with the second user request. The user profile data may be first user profile data, the user may be a first user, the user request may be a first user request, the network device may be a first network device and the data access request may be a first data access request, and the method may further include storing second user profile data corresponding to a second user profile in the first data repository, storing network device data corresponding to a second network device in the telecommunications network in a third data repository, detecting a second request from a second user for network device data corresponding to the second network device, where the second user request is associated with the second user profile, generating a second data access request to the third data repository utilizing the second user profile data from the first data repository and retrieving network device data from the third data repository in accordance with the second user request.

In another aspect, the present invention provides a method of managing a telecommunications network including storing user profile data corresponding to a user profile in a first data repository, wherein the user profile data includes a group name, storing network device data corresponding to a network device in the telecommunications network in a second data repository, detecting a request from a user for network device data corresponding to the network device, where the user request is associated with the user profile, generating a data access request to the second data repository utilizing the user profile data from the first data repository, where the data access request includes the group name, and retrieving network device data from the second data repository in accordance with the group name. The first data repository may be a central data repository and the second data repository may be embedded within the network device, and both the first and second data repositories may be databases. Retrieving network device data from the second data repository in accordance with the group name may include searching the second data repository for

the group name and retrieving network device data from the second data repository corresponding to the group name if the group name is found in the second data repository. The network device data retrieved from the second data repository may be configured resource data associated with the group name, and retrieving network device data from the second data repository corresponding to the group name may include using the group name to dynamically determine which data in the second data repository is retrieved. The first and second data repositories may be databases and retrieving network device data from the second database corresponding to the group name may include using the group name in a database query to actively filter which data in the second database is retrieved, and generating a data access request to the second database may include generating a where clause including the group name and sending the where clause to the second database. The user profile data may include group names and generating a data access request to the second data repository utilizing the user profile data from the first data repository may include generating a data access request to the second data repository for each group name in the user profile data, where each data access request includes the corresponding group name, and retrieving network device data from the second data repository in accordance with the group name may include retrieving network device data from the second data repository in accordance with each group name.

In yet another aspect, the present invention provides a method of managing a telecommunications network including storing user profile data corresponding to a user profile in a central data repository, detecting a user log-on request corresponding to the user profile through a network management system (NMS) and generating a user profile logical managed object (LMO) using the user profile data stored in the central data repository. The method may also include storing network device data in a network device data repository embedded within a network device in the telecommunications network, where the data corresponds to configured resources within the network device, detecting through the NMS a request for data corresponding to the configured resources within the network device from a user associated with the user profile and retrieving network device data from the network device data repository in accordance with the user profile data and the request for data. The user profile data may include one or more group names, where the user profile LMO includes the one or more group names, and retrieving network device

data from the network device data repository in accordance with the user profile data and the request for data may include searching the network device data repository for a match with each group name in the user profile LMO, returning an empty data set for each group name not found in the network device data repository and retrieving data corresponding to configured resources from the network device data repository for each group name found in the network device data repository. The central data repository and network device data repository may comprise databases.

Brief Description of the Drawings

Fig. 1 is a block diagram of a computer system with a distributed processing system;

Figs. 2a-2b are block and flow diagrams of a distributed network management system;

Fig. 3a is a block diagram of a logical system model;

Figs. 3b and 3d-3f are flow diagrams depicting a software build process using a logical system model;

Fig. 3c is a flow diagram illustrating a method for allowing applications to view data within a database;

Fig. 3g is a flow diagram depicting a configuration process;

Figs. 3h and 3j are flow diagrams depicting template driven network services provisioning processes;

Figs. 3i and 3k-3m are screen displays of an OSS client and various templates;

Figs. 4a-4z, 5a-5z, 6a-6p, 7a-7y, 8a-8e, 9a-9n, 10a-10i, 11a-11k, 11n-11o, 11s and 11x are screen displays of graphical user interfaces;

Figs. 11L-11m are tables representing data in a configuration database;

Figs. 11p-11r and 11t-11u are tables representing data in a network management system (NMS) database;

Fig. 11v is a block and flow diagram representing the creation of a user profile logical managed object including one or more groups;

Fig. 11w is a block and flow diagram of a network management system implementing user profiles and groups across multiple databases;

Figs. 12a and 13a are block and flow diagrams of a computer system incorporating a modular system architecture and illustrating a method for accomplishing hardware inventory and setup;

Figs. 12b-12c and 14a-14f are tables representing data in a configuration database;

Fig. 13b is a block and flow diagram of a computer system incorporating a modular system architecture and illustrating a method for configuring the computer system using a network management system;

Figs. 13c and 13d are block and flow diagrams of an accounting subsystem for pushing network device statistics to network management system software;

Fig. 15 is a block and flow diagram of a line card and a method for executing multiple instances of processes;

Figs. 16a-16b are flow diagrams illustrating a method for assigning logical names for inter-process communications;

Fig. 16c is a block and flow diagram of a computer system incorporating a modular system architecture and illustrating a method for using logical names for inter-process communications;

Fig. 16d is a chart representing a message format;

Figs. 17-19 are block and flow diagrams of a computer system incorporating a modular system architecture and illustrating methods for making configuration changes;

Fig. 20 is a block and flow diagram of a computer system incorporating a modular system architecture and illustrating a method for distributing logical model changes to users;

Fig. 21 is a block and flow diagram of a computer system incorporating a modular system architecture and illustrating a method for making a process upgrade;

Fig. 22 is a block diagram representing a revision numbering scheme;

Fig. 23 is a block and flow diagram of a computer system incorporating a modular system architecture and illustrating a method for making a device driver upgrade;

Fig. 24 is a block diagram representing processes within separate protected memory blocks;

Fig. 25 is a block and flow diagram of a line card and a method for accomplishing vertical fault isolation;

Fig. 26 is a block and flow diagram of a computer system incorporating a hierarchical and configurable fault management system and illustrating a method for accomplishing fault escalation.

Fig. 27 is a block diagram of an application having multiple sub-processes;

Fig. 28 is a block diagram of a hierarchical fault descriptor;

Fig. 29 is a block and flow diagram of a computer system incorporating a distributed redundancy architecture and illustrating a method for accomplishing distributed software redundancy;

Fig. 30 is a table representing data in a configuration database;

Figs. 31a-31c, 32a-32c, 33a-33d and 34a-34b are block and flow diagrams of a computer system incorporating a distributed redundancy architecture and illustrating methods for accomplishing distributed redundancy and recovery after a failure;

Fig. 35 is a block diagram of a network device;

Fig. 36 is a block diagram of a portion of a data plane of a network device;

Fig. 37 is a block and flow diagram of a network device incorporating a policy provisioning manager;

Figs. 38 and 39 are tables representing data in a configuration database;

Fig. 40 is an isometric view of a network device;

Figs. 41a-41c are front, back and side block diagrams, respectively, of components and modules within the network device of Fig. 40;

Fig. 42 is a block diagram of dual mid-planes;

Fig. 43 is a block diagram of two distributed switch fabrics and a central switch fabric;

Fig. 44 is a block diagram of the interconnections between switch fabric central timing subsystems and switch fabric local timing subsystems;

Fig. 45 is a block diagram of a switch fabric central timing subsystem;

Fig. 46 is a state diagram of master / slave selection for switch fabric central timing subsystems;

Fig. 47 is a block diagram of a switch fabric local timing subsystem;

Fig. 48 is a state diagram of reference signal selection for switch fabric local timing subsystems;

Fig. 49 is a block diagram of the interconnections between external central timing subsystems and external local timing subsystems;

Fig. 50 is a block diagram of an external central timing subsystem;

Fig. 51 is a timing diagram of a first timing reference signal with an embedded second timing signal;

Fig. 52 is a block diagram of an embeddor circuit;

Fig. 53 is a block diagram of an extractor circuit;

Fig. 54 is a block diagram of an external local timing subsystem;

Fig. 55 is a block diagram of an external central timing subsystem;

Fig. 56 is a block diagram of a network device connected to test equipment through programmable physical layer test ports;

Fig. 57 is a block and flow diagram of a network device incorporating programmable physical layer test ports;

Fig. 58 is a block diagram of a test path table;

Fig. 59 is a block and flow diagram of a network management system incorporating proxies to improve NMS server scalability;

Figs. 60a-60n are tables representing data in a configuration database;

Fig. 61a is a block diagram representing a physical managed object;

Fig. 61b is a block diagram representing a proxy; and

Fig. 62 is a screen display of a dialog box.

Detailed Description

A modular software architecture solves some of the more common scenarios seen in existing architectures when software is upgraded or new features are deployed. Software modularity involves functionally dividing a software system into individual modules or processes, which are then designed and implemented independently. Inter-process communication (IPC) between the processes is carried out through message passing in accordance with well-defined application programming interfaces (APIs) generated from the same logical system model using the same code generation system. A database process is used to maintain a primary data repository within the computer system / network device, and APIs for the database process are also generated from the same logical system model and using the same code generation system ensuring that all the processes access the same data in the same way. Another database process is used to maintain a secondary data repository external to the computer system / network device; this database receives all of its data by exact database replication from the primary database.

A protected memory feature also helps enforce the separation of modules. Modules are compiled and linked as separate programs, and each program runs in its own protected memory space. In addition, each program is addressed with an abstract communication handle, or logical name. The logical name is location-independent; it can live on any card in the system. The logical name is resolved to a physical

card/process during communication. If, for example, a backup process takes over for a failed primary process, it assumes ownership of the logical name and registers its name to allow other processes to re-resolve the logical name to the new physical card/process. Once complete, the processes continue to communicate with the same logical name, unaware of the fact that a switchover just occurred.

Like certain existing architectures, the modular software architecture dynamically loads applications as needed. Beyond prior architectures, however, the modular software architecture removes significant application dependent data from the kernel and minimizes the link between software and hardware. Instead, under the modular software architecture, the applications themselves gather necessary information (i.e., metadata and instance data) from a variety of sources, for example, text files, JAVA class files and database views, which may be provided at run time or through the logical system model.

Metadata facilitates customization of the execution behavior of software processes without modifying the operating system software image. A modular software architecture makes writing applications – especially distributed applications – more difficult, but metadata provides seamless extensibility allowing new software processes to be added and existing software processes to be upgraded or downgraded while the operating system is running. In one embodiment, the kernel includes operating system software, standard system services software and modular system services software. Even portions of the kernel may be hot upgraded under certain circumstances. Examples of metadata include, customization text files used by software device drivers; JAVA class files that are dynamically instantiated using reflection; registration and deregistration protocols that enable the addition and deletion of software services without system disruption; and database view definitions that provide many varied views of the logical system model. Each of these and other examples are described below.

The embodiment described below includes a network computer system with a loosely coupled distributed processing system. It should be understood, however, that the computer system could also be a central processing system or a combination of distributed and central processing and either loosely or tightly coupled. In addition,

the computer system described below is a network switch for use in, for example, the Internet, wide area networks (WAN) or local area networks (LAN). It should be understood, however, that the modular software architecture can be implemented on any network device (including routers) or other types of computer systems and is not restricted to a network switch.

A distributed processing system is a collection of independent computers that appear to the user of the system as a single computer. Referring to Fig. 1, computer system 10 includes a centralized processor 12 with a control processor subsystem 14 that executes an instance of the kernel 20 including master control programs and server programs to actively control system operation by performing a major portion of the control functions (e.g., booting and system management) for the system. In addition, computer system 10 includes multiple line cards 16a-16n. Each line card includes a control processor subsystem 18a-18n, which runs an instance of the kernel 22a-22n including slave and client programs as well as line card specific software applications. Each control processor subsystem 14, 18a-18n operates in an autonomous fashion but the software presents computer system 10 to the user as a single computer.

Each control processor subsystem includes a processor integrated circuit (chip) 24, 26a-26n, for example, a Motorola 8260 or an Intel Pentium processor. The control processor subsystem also includes a memory subsystem 28, 30a-30n including a combination of non-volatile or persistent (e.g., PROM and flash memory) and volatile (e.g., SRAM and DRAM) memory components. Computer system 10 also includes an internal communication bus 32 connected to each processor 24, 26a-26n. In one embodiment, the communication bus is a switched Fast Ethernet providing 100Mb of dedicated bandwidth to each processor allowing the distributed processors to exchange control information at high frequencies. A backup or redundant Ethernet switch may also be connected to each board such that if the primary Ethernet switch fails, the boards can fail-over to the backup Ethernet switch.

In this example, Ethernet 32 provides an out-of-band control path, meaning that control information passes over Ethernet 32 but the network data being switched by computer system 10 passes to and from external network connections 31a-31xx over a separate data path 34. External network control data is passed from the line cards to

the central processor over Ethernet 32. This external network control data is also assigned a high priority when passed over the Ethernet to ensure that it is not dropped during periods of heavy traffic on the Ethernet.

In addition, another bus 33 is provided for low level system service operations, including, for example, the detection of newly installed (or removed) hardware, reset and interrupt control and real time clock (RTC) synchronization across the system. In one embodiment, this is an Inter-IC communications (I^2C) bus.

Alternatively, the control and data may be passed over one common path (in-band).

Network/Element Management System (NMS):

Exponential network growth combined with continuously changing network requirements dictates a need for well thought out network management solutions that can grow and adapt quickly. The present invention provides a massively scalable, highly reliable comprehensive network management system, intended to scale up (and down) to meet varied customer needs.

Within a telecommunications network, element management systems (EMSs) are designed to configure and manage a particular type of network device (e.g., switch, router, hybrid switch-router), and network management systems (NMSs) are used to configure and manage multiple heterogeneous and/or homogeneous network devices. Hereinafter, the term "NMS" will be used for both element and network management systems. To configure a network device, the network administrator uses the NMS to provision services. For example, the administrator may connect a cable to a port of a network device and then use the NMS to enable the port. If the network device supports multiple protocols and services, then the administrator uses the NMS to provision these as well. To manage a network device, the NMS interprets data gathered by programs running on each network device relevant to network configuration, security, accounting, statistics, and fault logging and presents the interpretation of this data to the network administrator. The network administrator may use this data to, for example, determine when to add new hardware and/or services to the network device, to determine when new network devices should be added to the network, and to determine the cause of errors.

Preferably, NMS programs and programs executing on network devices perform in expected ways (i.e., synchronously) and use the same data in the same way. To avoid having to manually synchronize all integration interfaces between the various programs, a logical system model and associated code generation system are used to generate application programming interfaces (APIs) – that is integration interfaces / integration points -- for programs running on the network device and programs running within the NMS. In addition, the APIs for the programs managing the data repositories (e.g., database programs) used by the network device and NMS programs are also generated from the same logical system model and associated code generation system to ensure that the programs use the data in the same way. Further, to ensure that the NMS and network device programs for managing and operating the network device use the same data, the programs, including the NMS programs, access a single data repository for configuration information, for example, a configuration database within the network device.

Referring to Fig. 2a, in the present invention, the NMS 60 includes one or more NMS client programs 850a-850n and one or more NMS server programs 851a-851n. The NMS client programs provide interfaces for network administrators. Through the NMS clients, the administrator may configure multiple network devices (e.g., computer system 10, Fig. 1; network device 540, Fig. 35). The NMS clients communicate with the NMS servers to provide the NMS servers with configuration requirements from the administrator. In addition, the NMS server provides the NMS client with network device management information, which the client then makes available to the administrator. "Pushing" data from a server to multiple clients synchronizes the clients with minimal polling. Reduced polling means less management traffic on the network and more device CPU cycles available for other management task. Communication between the NMS client and server is done via Remote Method Invocation (RMI) over Transmission Control Protocol (TCP), a reliable protocol that ensures no data loss.

The NMS client and server relationship prevents the network administrator from directly accessing the network device. Since several network administrators may be

managing the network, this mitigates errors that may result if two administrators attempt to configure the same network device at the same time.

The present invention also includes a configuration relational database 42 within each network device and an NMS relational database 61 external to the network device. The configuration database program may be executed by a centralized processor card or a processor on another card (e.g., 12, Fig. 1; 542, Fig. 35) within the network device, and the NMS database program may be executed by a processor within a separate computer system (e.g., 62, Fig. 13b). The NMS server stores data directly in the configuration database via JAVA Database Connectivity (JDBC) over TCP, and using JDBC over TCP, the configuration database, through active queries, automatically replicates any changes to NMS database 61. By using JDBC and a relational database, the NMS server is able to leverage database transactions, database views, database journaling and database backup technologies that help provide unprecedented system availability. Relational database technology also scales well as it has matured over many years. An active query is a mechanism that enables a client to post a blocked SQL query for asynchronous notification by the database when data changes are made after the blocked SQL query was made.

Similarly, any configuration changes made by the network administrator directly through console interface 852 are made to the configuration database and, through active queries, automatically replicated to the NMS database. Maintaining a primary or master repository of data within each network device ensures that the NMS and network device are always synchronized with respect to the state of the configuration. Replicating changes made to the primary database within the network device to any secondary data repositories, for example, NMS database 61, ensures that all secondary data sources are quickly updated and remain in lockstep synchronization.

Instead of automatically replicating changes to the NMS database through active queries, only certain data, as configured by the network administrator, may be replicated. Similarly, instead of immediate replication, the network administrator may configure periodic replication. For example, data from the master embedded database (i.e., the configuration database) can be uploaded daily or hourly. In

addition to the periodic, scheduled uploads, backup may be done anytime at the request of the network administrator.

Referring again to Fig. 2a, for increased availability, the network device may include a backup configuration database 42' maintained by a separate, backup centralized processor card (e.g., 12, Fig. 1; 543, Fig. 35). Any changes to configuration database 42 are replicated to backup configuration database 42'. If the primary centralized processor card experiences a failure or error, the backup centralized processor card may be switched over to become the primary processor and configuration database 42' may be used to keep the network device operational. In addition, any changes to configuration database 42 may be written immediately to flash persistent memory 853 which may also be located on the primary centralized processor card or on another card, and similarly, any changes to backup configuration database 42' may be written immediately to flash persistent memory 853' which may also be located on the backup centralized processor card or another card. These flash-based configuration files protect against loss of data during power failures. In the unlikely event that all copies of the database within the network device are unusable, the data stored in the NMS database may be downloaded to the network device.

Instead of having a single central processor card (e.g., 12, Fig. 1; 543, Fig. 35), the external control functions and the internal control functions may be separated onto different cards as described in U.S. Patent Application Serial Number ___, filed May 20, 2000 and entitled "Functional Separation of Internal and External Controls in Network Devices", which is hereby incorporated herein by reference. As shown in Figs. 41a and 41b, the chassis may support internal control (IC) processor cards 542a and 543a and external control (EC) processor cards 542b and 543b. In this embodiment, configuration database 42 may be maintained by a processor on internal control processor card 542a and configuration database 42' may be maintained by a processor on internal control processor card 543a, and persistent memory 853 may be located on external control processor card 542b and persistent memory 853' may be located on external control processor card 543b. This increases inter-card communication but also provides increased fault tolerance.

The file transfer protocol (FTP) may provide an efficient, reliable transport out of the network device for data intensive operations. Bulk data applications include accounting, historical statistics and logging. An FTP push (to reduce polling) may be used to send accounting, historical statistics and logging data to a data collector server 857, which may be a UNIX server. The data collector server may then generate network device and/or network status reports 858a-858n in, for example, American Standard Code for Information Interchange (ASCII) format and store the data into a database or generate Automatic Message Accounting Format (AMA/BAF) outputs.

Selected data stored within NMS database 61 may also be replicated to one or more remote/central NMS databases 854a-854n, as described below. NMS servers may also access network device statistics and status information stored within the network device using SNMP (multiple versions) traps and standard Management Information Bases (MIBs and MIB-2). The NMS server augments SNMP traps by providing them over the conventional User Datagram Protocol (UDP) as well as over Transmission Control Protocol (TCP), which provides reliable traps. Each event is generated with a sequence number and logged by the data collector server in a system log database for in place context with system log data. These measures significantly improve the likelihood of responding to all events in a timely manner reducing the chance of service disruption.

The various NMS programs – clients, servers, NMS databases, data collector servers and remote NMS databases – are distributed programs and may be executed on the same computer or different computers. The computers may be within the same LAN or WAN or accessible through the Internet. Distribution and hierarchy are fundamental to making any software system scale to meet larger needs over time. Distribution reduces resource locality constraints and facilitates flexible deployment. Since day-to-day management is done in a distributed fashion, it makes sense that the management software should be distributed. Hierarchy provides natural boundaries of management responsibility and minimizes the number of entities that a management tool must be aware of. Both distribution and hierarchy are fundamental to any long-term management solution. The client server model allows for increased scalability as servers and clients may be added as the number of network managers increase and as the network grows.

The various NMS programs may be written in the JAVA programming language to enable the programs to run on both Windows/NT and UNIX platforms, such as Sun Solaris. In fact the code for both platforms may be the same allowing consistent graphical interfaces to be displayed to the network administrator. In addition to being native to JAVA, RMI is attractive as the RMI architecture includes (RMI) over Internet Inter-Orb Protocol (IIOP) which delivers Common Object Request Broker Architecture (CORBA) compliant distributed computing capabilities to JAVA. Like CORBA, RMI over IIOP uses IIOP as its communication protocol. IIOP eases legacy application and platform integration by allowing application components written in C++, SmallTalk, and other CORBA supported languages to communicate with components running on the JAVA platform. For "manage anywhere" purposes and web technology integration, the various NMS programs may also run within a web browser. In addition, the NMS programs may integrate with Hewlett Packard's (HP's) Network Node Manager (NNM™) to provide the convenience of a network map, event aggregation/filtering, and integration with other vendor's networking. From HP NNM a context-sensitive launch into an NMS server may be executed.

The NMS server also keeps track of important statistics including average client/server response times and response times to each network device. By looking at these statistics over time, it is possible for network administrators to determine when it is time to grow the management system by adding another server. In addition, each NMS server gathers the name, IP address and status of other NMS servers in the telecommunication network, determines the number of NMS clients and network devices to which it is connected, tracks its own operation time, the number of transactions it has handled since initialization, determines the "top talkers" (i.e., network devices associated with high numbers of transactions with the server), and the number of communications errors it has experienced. These statistics help the network administrator tune the NMS to provide better overall management service.

NMS database 61 may be remote or local with respect to the network device(s) that it is managing. For example, the NMS database may be maintained on a computer system outside the domain of the network device (i.e., remote) and communications

between the network device and the computer system may occur over a wide area network (WAN) or the Internet. Preferably, the NMS database is maintained on a computer system within the same domain as the network device (i.e., local) and communications between the network device and the computer system may occur over a local area network (LAN). This reduces network management traffic over a WAN or the Internet.

Many telecommunications networks include domains in various geographical locations, and network managers often need to see data combined from these different domains to determine how the overall network is performing. To assist with the management of wide spread networks and still minimize the network management traffic sent over WANs and the Internet, each domain may include an NMS database 61 and particular / selected data from each NMS database may be replicated (or "rolled up") to remote NMS databases 854a-854n that are in particular centralized locations. Referring to Fig. 2b, for example, a telecommunications network may include at least three LAN domains 855a-855c where each domain includes multiple network devices 540 and an NMS database 61. Domain 855a may be located in the Boston, Massachusetts area, domain 855b may be located in the Chicago, Illinois area and domain 855c may be located in the San Francisco, California area. NMS servers 851a-851f may be located within each domain or in a separate domain. Similarly, one or more NMS clients may be coupled to each NMS server and located in the same domain as the NMS server or in different domains. In addition, one NMS client may be coupled with multiple NMS servers. For example, NMS servers 851a-851c and NMS clients 850a-850k may be located in domain 856a (e.g., Dallas, Texas) while NMS servers 851d-851f and NMS clients 850m-850u may be located in domain 856b (e.g., New York, New York). Each NMS server may be used to manage each domain 855a-855c or, preferably, one NMS server in each server domain 856a-856b is used to manage all of the network devices within one network device domain 855a-855c. A single domain may include network devices and NMS clients and servers.

Network administrators use the NMS clients to configure network devices in each of the domains through the NMS servers. The network devices replicate changes made to their internal configuration databases (42, Fig. 2a) to a local NMS database 61. In addition, the data collector server copies all logging data into NMS database 61 or a

separate logging database (not shown). Each local NMS database may also replicate selected data to central NMS database(s) 854a-854n in accordance with instructions from the network administrator. Other programs may then access the central database to retrieve and combine data from multiple network devices in multiple domains and then present this data to the network administrator. Importantly, network management traffic over WANs and the Internet are minimized since all data is not copied to the central NMS database. For example, local logging data may only be stored in the local NMS databases 61 (or local logging database) and not replicated to one of the central NMS database.

Logical System Model:

As previously mentioned, to avoid having to manually synchronize all integration interfaces between the various programs, the APIs for both NMS and network device programs are generated using a code generation system from the same logical system model. In addition, the APIs for the data repository software used by the programs are also generated from the same logical system model to ensure that the programs use the data in the same way. Each model within the logical system model contains metadata defining an object / entity, attributes for the object and the object's relationships with other objects. Upgrading / modifying an object is, therefore, much simpler than in current systems, since the relationship between objects, including both hardware and software, and attributes required for each object are clearly defined in one location. When changes are made, the logical system model clearly shows what other programs are affected and, therefore, may also need to be changed. Modeling the hardware and software provides a clean separation of function and form and enables sophisticated dynamic software modularity.

A code generation system uses the attributes and metadata within each model to generate the APIs for each program and ensure lockstep synchronization. The logical model and code generation system may also be used to create test code to test the network device programs and NMS programs. Use of the logical model and code generation system saves development, test and integration time and ensures that all relationships between programs are in lockstep synchronization. In addition, use of the logical model and code generation system facilitates hardware portability, seamless extensibility and unprecedented availability and modularity.

Referring to Fig. 3a, a logical system model 280 is created using the object modeling notation and a model generation tool, for example, Rational Rose 2000 Modeler Edition available from Rational Software Corporation in Lexington, Massachusetts. A managed device 282 represents the top level system connected to models representing both hardware 284 and data objects used by software applications 286. Hardware model 284 includes models representing specific pieces of hardware, for example, chassis 288, shelf 290, slot 292 and printed circuit board 294. The logical model is capable of showing containment, that is, typically, there are many shelves per chassis (1:N), many slots per shelf (1:N) and one board per slot (1:1). Shelf 290 is a parent class generalizing multiple shelf models, including various functional shelves 296a-296n as well as one or more system shelves, for example, for fans 298 and power 300. Board 294 is also a parent class having multiple board models, including various functional boards without external physical ports 302a-302n (e.g., central processor 12, Fig. 1; 542-543, Fig. 35; and switch fabric cards, Fig. 35) and various functional boards 304a-304n (e.g., cross connection cards 562a-562b and forwarding cards 546a-546e, Fig. 35) that connect to boards 306 with external physical ports (e.g., universal port cards 554a-554h, Fig. 35). Hardware model 284 also includes an external physical port model 308. Port model 308 is coupled to one or more specific port models, for example, synchronous optical network (SONET) protocol port 310, and a physical service endpoint model 312.

Hardware model 284 includes models for all hardware that may be available on computer system 10 (Fig. 1) / network device 540 (Fig. 35) whether a particular computer system / network device uses all the available hardware or not. The model defines the metadata for the system whereas the presence of hardware in an actual network device is represented in instance data. All shelves and slots may not be populated. In addition, there may be multiple chassis. It should be understood that SONET port 310 is an example of one type of port that may be supported by computer system 10. A model is created for each type of port available on computer system 10, including, for example, Ethernet, Dense Wavelength Division Multiplexing (DWDM) or Digital Signal, Level 3 (DS3). The NMS (described below) uses the hardware model and instance data to display a graphical picture of computer system 10 / network device 540 to a user.

Service endpoint model 314 spans the software and hardware models within logical model 280. It is a parent class including a physical service endpoint model 312 and a logical service endpoint model 316. Since the links between the software model and hardware model are minimal, either may be changed (e.g., upgraded or modified) and easily integrated with the other. In addition, multiple models (e.g., 280) may be created for many different types of managed devices (e.g., 282). The software model may be the same or similar for each different type of managed device even if the hardware – and hardware models – corresponding to the different managed devices are very different. Similarly, the hardware model may be the same or similar for different managed devices but the software models may be different for each. The different software models may reflect different customer needs.

Software model 286 includes models of data objects used by each of the software processes (e.g., applications, device drivers, system services) available on computer system 10 / network device 540. All applications and device drivers may not be used in each computer system / network device. As one example, ATM model 318 is shown. It should be understood that software model 286 may also include models for other applications, for example, Internet Protocol (IP) applications, Frame Relay and Multi-Protocol Label Switching (MPLS) applications. Models of other processes (e.g., device drivers and system services) are not shown for convenience.

For each process, models of configurable objects managed by those processes are also created. For example, models of ATM configurable objects are coupled to ATM model 318, including models for a soft permanent virtual path (SPVP) 320, a soft permanent virtual circuit (SPVC) 321, a switch address 322, a cross-connection 323, a permanent virtual path (PVP) cross-connection 324, a permanent virtual circuit (PVC) cross-connection 325, a virtual ATM interface 326, a virtual path link 327, a virtual circuit link 328, logging 329, an ILMI reference 330, PNNI 331, a traffic descriptor 332, an ATM interface 333 and logical service endpoint 316. As described above, logical service endpoint model 316 is coupled to service endpoint model 314. It is also coupled to ATM interface model 333.

The logical model is layered on the physical computer system to add a layer of abstraction between the physical system and the software applications. Adding or removing known (i.e., not new) hardware from the computer system will not require changes to the logical model or the software applications. However, changes to the physical system, for example, adding a new type of board, will require changes to the logical model. In addition, the logical model is modified when new or upgraded processes are created. Changes to an object model within the logical model may require changes to other object models within the logical model. It is possible for the logical model to simultaneously support multiple versions of the same software processes (e.g., upgraded and older). In essence, the logical model insulates software applications from changes to the hardware models and vice-versa.

To further decouple software processes from the logical model – as well as the physical system – another layer of abstraction is added in the form of version-stamped views. A view is a logical slice of the logical model and defines a particular set of data within the logical model to which an associated process has access. Version stamped views allow multiple versions of the same process to be supported by the same logical model since each version-stamped view limits the data that a corresponding process "views" or has access to, to the data relevant to the version of that process. Similarly, views allow multiple different processes to use the same logical model.

Code Generation System:

Referring to Fig. 3b, logical model 280 is used as input to a code generation system 336. The code generation system creates a view identification (id) and an application programming interface (API) 338 for each process that requires configuration data. For example, a view id and an API may be created for each ATM application 339a-339n, each SONET application 340a-340n, each MPLS application 342a-342n and each IP application 341a-341n. In addition, a view id and API is also created for each device driver process, for example, device drivers 343a-343n, and for modular system services (MSS) 345a-345n (described below), for example, a Master Control Driver (MCD), a System Resiliency Manager (SRM), and a Software Management System (SMS). The code generation system provides data consistency across processes, centralized tuning and an abstraction of embedded configuration and NMS databases

(described below) ensuring that changes to their database schema (i.e., configuration tables and relationships) do not affect existing processes.

The code generation system also creates a data definition language (DDL) file 344 including structured query language (SQL) commands used to construct the database schema, that is, the various tables and views within a configuration database 346, and a DDL file 348 including SQL commands used to construct various tables and SQL views within a network management (NMS) database 350 (described below). This is also referred to as converting the logical model into a database schema and various SQL views look at particular portions of that schema within the database. If the same database software is used for both the configuration and NMS databases, then one DDL file may be used for both.

The databases do not have to be generated from a logical model for views to work. Instead, database files can be supplied directly without having to generate them using the code generation system. Similarly, instead of using a logical model as an input to the code generation system, a MIB "model" may be used. For example, relationships between various MIBs and MIB objects may be written (i.e., coded) and then this "model" may be used as input to the code generation system.

Referring to Fig. 3c, applications 352a-352n (e.g., SONET driver 863, SONET application 860, MSS 866, etc.) each have an associated view 354a-354n of configuration database 42. The views may be similar allowing each application to view similar data within configuration database 42. For example, each application may be ATM version 1.0 and each view may be ATM view version 1.3. Instead, the applications and views may be different versions. For example, application 352a may be ATM version 1.0 and view 354a may be ATM view version 1.3 while application 352b is ATM version 1.7 and view 354b is ATM view version 1.5. A later version, for example, ATM version 1.7, of the same application may represent an upgrade of that application and its corresponding view allows the upgraded application access only to data relevant to the upgraded version and not data relevant to the older version. If the upgraded version of the application uses the same configuration data as an older version, then the view version may be the same for both applications. In addition, application 352n may represent a completely different type of application,

for example, MPLS, and view 354n allows it to have access to data relevant to MPLS and not ATM or any other application. Consequently, through the use of database views, different versions of the same software applications and different types of software applications may be executed on computer system 10 simultaneously.

Views also allow the logical model and physical system to be changed, evolved and grown to support new applications and hardware without having to change existing applications. In addition, software applications may be upgraded and downgraded independent of each other and without having to re-boot computer system 10 / network device 540. For example, after computer system 10 is shipped to a customer, changes may be made to hardware or software. For instance, a new version of an application, for example, ATM version 2.0, may be created or new hardware may be released requiring a new or upgraded device driver process. To make this a new process and/or hardware available to the user of computer system 10, first the software image including the new process must be re-built.

Referring again to Fig. 3b, logical model 280 may be changed (280') to include models representing the new software and/or hardware. Code generation system 336 then uses new logical model 280' to re-generate view ids and APIs 338' for each application, including, for example, ATM version two 360 and device driver 362, and DDL files 344' and 348'. The new application(s) and/or device driver(s) processes then bind to the new view ids and APIs. A copy of the new application(s) and/or device driver process as well as the new DDL files and any new hardware are sent to the user of computer system 10. The user can then download the new software and plug the new hardware into computer system 10. The upgrade process is described in more detail below. Similarly, if models are upgraded / modified to reflect upgrades / modifications to software or hardware, then the new logical model is provided to the code generation system which re-generates view ids and APIs for each process / program / application. Again, the new applications are linked with the new view ids and APIs and the new applications and/or hardware are provided to the user.

Again referring to Fig. 3b, the code generation system also creates NMS JAVA interfaces 347 and persistent layer metadata 349. The JAVA interfaces are JAVA class files including get and put methods corresponding to attributes within the logical

model, and as described below, the NMS servers use the NMS JAVA interfaces to construct models of each particular network device to which they are connected. Also described below, the NMS servers use the persistent layer metadata as well as run time configuration data to generate SQL configuration commands for use by the configuration database.

Prior to shipping computer system 10 to customers, a software build process is initiated to establish the software architecture and processes. The code generation system is the first part of this process. Following the execution of the code generation system, each process when pulled into the build process links the associated view id and API into its image. For example, referring to Fig. 3d, to build a SONET application, source files, for example, a main application file 859a, a performance monitoring file 859b and an alarm monitoring file 859c, written in, for example, the C programming language (.c) are compiled into object code files (.o) 859a', 859b' and 859c'. Alternatively, the source files may be written in other programming languages, for example, JAVA (.java) or C++ (.cpp). The object files are then linked along with view ids and APIs from the code generation system corresponding to the SONET application, for example, SONET API 340a. The SONET API may be a library (.a) of many object files. Linking these files generates the SONET Application executable file (.exe) 860.

Referring to Fig. 3e, each of the executable files for use by the network device / computer system are then provided to a kit builder 861. For example, several SONET executable files (e.g., 860, 863), ATM executable files (e.g., 864a-864n), MPLS executable files (e.g., 865a-865n), MSS executable files 866a-866n and a DDL configuration database executable file 867 may be provided to kit builder 861. Alternatively, the DDL configuration database executable file may be executed and some data placed in the database prior to supplying the DDL file to the kit builder. The kit builder creates a computer system / network device installation kit 862 that is shipped to the customer with the computer system / network device or, later, alone after modifications and upgrades are made.

Referring to Fig. 3f, similarly, each of the executable files for the NMS is provided separately to the kit builder. For example, a DDL NMS database executable file 868,

an NMS JAVA interfaces executable file 869, a persistent layer metadata executable file 870, an NMS server 885 and an NMS client 886 may be provided to kit builder 861. The kit builder creates an NMS installation kit 871 that is shipped to the customer for installation on a separate computer 62 (Fig. 13b). In addition, new versions of the NMS installation kit may be sent to customers later after upgrades / modifications are made. When installing the NMS, the customer / network administrator may choose to distribute the various NMS processes as described above. Alternatively, one or more of the NMS programs, for example, the NMS JAVA interfaces and Persistent layer metadata executable files may be part of the network device installation kit and later passed from the network device to the NMS server, or part of both the network device installation kit and the NMS installation kit.

When the computer system is powered-up for the first time, as described below, configuration database software uses DDL file 867 to create a configuration database 42 with the necessary configuration tables and active queries. The NMS database software uses DDL file 868 to create NMS database 61 with corresponding configuration tables. Memory and storage space within network devices is typically very limited. The configuration database software is robust and takes a considerable amount of these limited resources but provides many advantages as described below.

As described above, logical model 280 (Fig. 3b) may be provided as an input to code generation system 336 in order to generate database views and APIs for NMS programs and network device programs to synchronize the integration interfaces between those programs. Where a telecommunications network includes multiple similar network devices, the same installation kit may be used to install software on each network device to provide synchronization across the network. Typically, however, networks include multiple different network devices as well as multiple similar network devices. A logical model may be created for each different type of network device and a different installation kit may be implemented on each different type of network device.

Instead, of providing a logical model (e.g., 280, Fig. 3b) that represents a single network device, a logical model may be provided that represents multiple different managed devices – that is, multiple network devices and the relationship between the

network devices. Alternatively, multiple logical models 280 and 887a-887n – representing multiple network devices -- may be provided, including relationships with other logical models. In either case, providing multiple logical models or one logical model representing multiple network devices and their relationships as an input(s) to the code generation system allows for synchronization of NMS programs and network device programs (e.g., 901a-901n) across an entire network. The code generation system in combination with one or more logical models provides a powerful tool for synchronizing distributed telecommunication network applications.

The logical model or models may also be used for simulation of a network device and/or a network of many network devices, which may be useful for scalability testing.

In addition to providing view ids and APIs, the code generation system may also provide code used to push data directly into a third party code API. For example, where an API of a third party program expects particular data, the code generation system may provide this data by retrieving the data from the central repository and calling the third-party programs API. In this situation, the code generation system is performing as a "data pump".

Configuration:

Once the network device programs have been installed on network device 540 (Fig. 35), and the NMS programs have been installed on one or more computers (e.g., 62), the network administrator may configure the network device / provision services within the network device. Hereinafter, the term "configure" includes "provisioning services". Referring to Fig. 4a, the NMS client displays a graphical user interface (GUI) 895 to the administrator including a navigation tree / menu 898. Selecting a branch of the navigation tree causes the NMS client to display information corresponding to that branch. For example, selecting Devices branch 898a within the tree causes the NMS client to display a list 898b of IP addresses and/or domain name server (DNS) names corresponding to network devices that may be managed by the

administrator. The list corresponds to a profile associated with the administrator's user name and password. Profiles are described in detail below.

If the administrator's profile includes the appropriate authority, then the administrator may add new devices to list 898b. To add a new device, the administrator selects Devices branch 898a and clicks the right mouse button to cause a pop-up menu 898c (Fig. 4b) to appear. The administrator then selects the Add Devices option to cause a dialog box 898d (Fig. 4c) to appear. The administrator may then type in an IP address (e.g., 192.168.9.203) or a DNS name into field 898e and select an Add button 898f to add the device to Device list window 898g (Fig. 4d). The administrator may then add one or more other devices in a similar manner. The administrator may also delete a device from the Device list window by selecting the device and then selecting a Delete button 898h, or the administrator may cancel out of the dialog box without adding any new devices by selecting Cancel button 898i. When finished, the administrator may select an OK button 898j to add any new devices in Device list 898g to navigation tree 898a (Fig. 4e).

To configure a network device, the administrator begins by selecting (step 874, Fig. 3g) a particular network device to configure, for example, the network device corresponding to IP address 192.168.9.202 (Fig. 4f). The NMS client then informs (step 875, Fig. 3g) an NMS server of the particular network device to be configured. Since many NMS clients may connect to the same NMS server, the NMS server first checks its local cache to determine if it is already managing the network device for another NMS client. If so, the NMS server sends data from the cache to the NMS client. If not, the NMS server using JDBC connects to the network device and reads the data / object structure for the physical aspects of the device from the configuration database within the network device into its local cache and uses that information with the JAVA interfaces to construct (step 876) a model of the network device. The server provides (step 877) this information to the client, which displays (step 878) a graphical representation 896a (Fig. 4f) of the network device to the administrator indicating the hardware and services available in the selected network device and the current configuration and currently provisioned services. Configuration changes received by an NMS server – from either an NMS client or directly from the network device's configuration database when changes are made through the network device's

CLI interface -- are sent by the NMS server to any other NMS clients connected to that server and managing the same network device. This provides scalability, since the device is not burdened with multiple clients subscribing for traps, and ensures each NMS client provides an accurate view of the network device.

Referring to Figs. 4f-4l, graphical representation 896a (i.e., device view, device mimic) in graphic window 896b may include many views of the network device. For example, device mimic 896a is shown in Fig. 4f displaying a front view of the components in the upper portion of network device 540 (Fig. 35). The administrator may use scroll bar 926a to scroll down and view lower portions of the front of the network device as shown in Fig. 4g. The administrator may also use image scale button 926b to change the size of graphic 896a. For example, the administrator may shrink the network device image to allow more of the device image to be visible in graphic window 896b, as shown in Fig. 4h. This view corresponds to the block diagram of network device 540 shown in Fig. 41a. For instance, upper fan tray 634 and middle fan trays 630 and 632 are shown. In addition, forwarding cards (e.g., 546a and 548e), cross-connection cards (e.g., 562a, 562b, 564b, 566a, 568b), and external processor control cards (e.g., 542b and 543b) are shown.

GUI 895 also includes several splitter bars 895a-895c (Fig. 4f) to allow the administrator to change the size of the various panels (e.g., 896b, 897 and 898). In addition, GUI 895 includes a status bar 895d. The status bar may include various fields such as a server field 895e, a Mode field 895f, a Profile field 895g and an active field 895h. The server filed may provide the IP address or DNS name of the NMS server, and the profile field may provide the username that the administrator logged in under. The active field will provide updated status, for example, ready, or ask the administrator to take particular steps. The mode field will indicate an on-line mode (i.e., typical operation) or an off-line mode (described in detail below).

Device mimic 896a may also provide one or more visual indications as to whether a card is present in each slot or whether a slot is empty. For example, in one embodiment, the forwarding cards (e.g., 546a and 548e) in the upper portion of the network device are displayed in a dark color to indicate the cards are present while the lower slots (e.g., 928a and 929e) are shown in a lighter color to indicate that the slots

are empty. Other visual indications may also be used. For example, a graphical representation of the actual card faceplate may be added to device mimic 896a when a card is present and a blank faceplate may be added when the slot is empty. Moreover, this may be done for any of the cards that may or may not be present in a working network device. For example, the upper cross-connection cards may be displayed in a dark color to indicate they are present while the lower cross-connection card slots may be displayed in a lighter color to indicate the slots are empty.

In addition, a back view and other views of the network device may also be shown. For example, the administrator may use a mouse to move a cursor into an empty portion of graphic window 896b and click the right mouse button to cause a pop-up menu to appear listing the various views available for the network device. In one embodiment, the only other view is a back view and pop-up menu 927 is displayed. Alternatively, short cuts may be set up. For example, double clicking the left mouse button may automatically cause graphic 896a to display the back view of the network device, and another double click may cause graphic 896a to again display the front view. As another alternative, a pull down menu may be provided to allow an administrator to select between various views.

Device mimic 896a is shown in Fig. 4i displaying a back view of the components in the upper portion of network device 540 (Fig. 35). Again the administrator may use scroll bar 926a and/or image scale button 926b to view lower portions (Figs. 4j and 4k) of the back of the network device or more of the network device by shrinking the graphic (Fig. 4l). These views correspond to the block diagram of network device 540 shown in Fig. 41b. For example, upper fan tray 628 (Fig. 4i), management interface (MI) card 621 (Fig. 4i) and lower fan tray 626 (Fig. 4k) are shown. In addition, universal port cards (e.g., 556h, 554a and 560h, Fig. 4l), switch fabric cards (e.g., 570a and 570b) and internal processor control cards (e.g., 542a and 543a) are also shown. Again, graphic 896a may use a visual indicator to clearly show whether a card is present in a slot or whether the slot is empty. In this example, the visual indicator for universal port cards is the display of the ports available on each card. For example, universal port card 554a is present as indicated by the graphical representation of ports (e.g., 930, Fig. 4l) available on that card, while universal port card 558a (Fig. 41b) is not present as indicated by a blank slot 931.

Since the GUI has limited screen real estate and the network device may be large and loaded with many different types of components (e.g., modules, ports, fan trays, power connections), in addition to the device mimic views described above, GUI 895 may also provide a system view menu option 954 (Fig. 4m). If an administrator selects this option, a separate pull away window 955 (Fig. 4n) is displayed for the administrator including both a front view 955a and a back view 955b of the network device corresponding to the front and back views displayed by the device mimic. The administrator may keep this separate pull away window up and visible while provisioning services through the GUI. Moreover, the GUI remains linked with the pull away window such that if the administrator selects a component in the pull away window, the device mimic displays that portion of the device and highlights that component. Similarly, if the administrator selects a component within the device mimic, the pull away window also highlights the selected component. Thus, the pull away window may further help the administrator navigate in the device mimic.

Device mimic 896a may also indicate the status of components. For example, ports and/or cards may be green for normal operation, red if there are errors and yellow if there are warnings. In one embodiment, a port may be colored, for example, light green or gray if it is available but not yet configured and colored dark green after being configured. Other colors or graphical textures may also be used show visible status. To further ease a network administrator's tasks, the GUI may present pop-up windows or tool tips containing information about each card and/or port when the administrator moves the cursor over the card or port. For example, when the administrator moves the cursor over universal port card 556f (Fig. 4o), pop-up window 932a may be displayed to tell the administrator that the card is a 16 Port OC3 Universal Port Module in Shelf 11 / Slot 3. Similarly, if the administrator moves the cursor over universal port card 556e (Fig. 4p), pop-up window 932b appears indicating that the card is a 16 Port OC12 Universal Port Module in Shelf 11 / Slot 4, and if the cursor is moved over universal port cards 556d (Fig. 4q) or 556c (Fig. 4r), then pop-up windows 932c and 932d appear indicating the cards are 4 Port OC48 Universal Port Module in Shelf 11 / Slot 5 and 8 Port OC12 Universal Port Module in Shelf 11 / Slot 6, respectively. If the administrator moves the cursor over a port, for

example, port 933 (Fig. 4s), then pop-up window 932e appears indicating the port is an OC12 in Shelf 11 / Slot 4 / Port 1.

The views are used to provide management context. The GUI may also include a configuration / service status window 897 for displaying current configuration and service provisioning details. Again, these details are provided to the NMS client by the NMS server, which reads the data from the network device's configuration database. The status window may include many tabs / folders for displaying various data about the network device configuration. In one embodiment, the status window includes a System tab 934 (Fig. 4s), which is displayed when the server first accesses the network device. This tab provides system level data such as the system name 934a, System Description 934b, System Contact 934c, System Location 934d, System IP Address 934e (or DNS name), System Up Time 934f, System identification (ID) 934g and System Services 934h. Modifications to data displayed in 934a-934e may be made by the administrator and committed by selecting the Apply button 935. The NMS client then passes this information to the NMS server, which then writes a copy of the data in the network device's configuration database and broadcasts the changes to any other NMS clients managing the same network device. The administrator may also reset the network device by selecting the Reset System button 935b and then refresh the System tab data by selecting the Refresh button 935c.

The status window may also include a Modules tab 936 (Fig. 4t), which includes an inventory of the available modules in the network device and various details about those modules such as where they are located (e.g., shelf and slot, back or front). The inventory may also include a description of the type of module, version number, manufacturing date, part number, etc. In addition, the inventory may include run time data such as the operational status and temperature. The NMS server may continuously supply the NMS client(s) with the run time data by reading the network device configuration database or NMS database. Device mimic 896a is linked with status window 897, such that selecting a module in device mimic 896a causes the Module tab to highlight a line in the inventory corresponding to that card. For example, if an administrator selects universal port card 556d, device mimic 896a highlights that module and the Module tab highlights a line 937 in the inventory corresponding to the card in Shelf 11 / Slot 5. Similarly, if the administrator selects a

line in the Module tab inventory, device mimic 896a highlights the corresponding module. Double clicking the left mouse button on a selected module may cause a dialog box to appear and the administrator may modify particular parameters such as an enable/disable parameter.

The status window may also include a Ports tab 938 (Fig. 4u), which displays an inventory of the available ports in the network device and various details about each port such as where they are located (shelf, slot and port; back or front). The inventory may also include a description of the port name, type and speed as well as run time data such as administrative status, operational status and link status. Again, device mimic 896a is linked with status window 897 such that selecting a port within device mimic 896a causes the Port tab to highlight a line in the inventory corresponding to that port. For example, if the administrator selects port 939a (port 1, slot 4) on card 556e, then the Port tab highlights a line 939b within the inventory corresponding to that port. Similarly, if the administrator selects a line from the inventory in the Port tab, device mimic 896a highlights the corresponding port. Again double clicking the left mouse button on a selected port may cause a dialog box to appear and the administrator may modify particular parameters such as an enable/disable parameter.

Another tab in the status window may be a SONET Interface tab 940 (Fig. 4v), which includes an inventory of SONET ports in the network device and various details about each port such as where they are located (shelf and slot; back or front). Medium type (e.g., SONET, Synchronous Digital Hierarchy (SDH)) may also be displayed as well as circuit ID, Line Type, Line Coding, Loopback, Laser Status, Path Count and other details. Again, device mimic 896a is lined with status window 897 such that selecting a port within device mimic 896a causes the SONET Interface tab to highlight a line in the inventory corresponding to that SONET port. For example, if the administrator selects port 941a (port 2, slot 5) on card 556d, then the SONET Interface tab highlights line 941b corresponding to that port. Similarly, if the administrator selects a line from the inventory in the SONET Interface tab, device mimic 896a highlights the corresponding port. Again, double clicking the left mouse button on a selected SONET interface may cause a dialog box to appear and the administrator may modify particular parameters such as an enable/disable parameter.

The System tab data as well as the Modules tab, Ports tab and SONET Interface tab data all represent physical aspects of the network device. The remaining tabs, including SONET Paths tab 942 (Fig. 4w), ATM Interfaces tab 946, Virtual ATM Interfaces tab 947 and Virtual Connections tab 948, display configuration details and, thus, display no data until the device is configured. In addition, these configuration tabs 942, 946-948 are dialog chained together with wizard-like properties to guide an administrator through configuration details. Through these tabs within the GUI (i.e., graphical context), therefore, the administrator then makes (step 879, Fig. 3g) configuration selections. For example, to configure a SONET path, the administrator may begin by selecting a port (e.g., 939a on card 556e, Fig. 5a) within device mimic 896a and clicking the right mouse button (i.e., context sensitive) to cause a pop-up menu 943 to be displayed listing available port configuration options. The administrator may then select the "Configure SONET Paths" option, which causes the GUI to display a SONET Path configuration wizard 944 (Fig. 5b).

The SONET Path configuration wizard guides the administrator through the task of setting up a SONET Path by presenting the administrator with valid configuration options and inserting default parameter values. As a result, the process of configuring SONET paths is simplified, and required administrator expertise is reduced since the administrator does not need to know or remember to provide each parameter value. In addition, the SONET Path wizard allows the administrator to configure multiple SONET Paths simultaneously, thereby eliminating the repetition of similar configuration process steps required by current network management systems and reducing the time required to configure many SONET Paths. Moreover, the wizard validates configuration requests from the administrator to minimize the potential for mis-configuration.

In one embodiment, the SONET Path wizard displays SONET line data 944a (e.g., slot 4, port 1, OC12) and three configuration choices 944b, 944c and 944d. The first two configuration choices provide "short cuts" to typical configurations. If the administrator selects the first configuration option 944b (Fig. 5c), the SONET Path wizard creates a single concatenated path. In the current example, the selected port is an OC12, and the single concatenated path is an STS-12c. The wizard assigns and graphically displays the position 944e and width 944f of the STS-12c path and also

displays a SONET Path table 944g including an inventory having an entry for the SONET STS-12c path and each of the default parameters assigned to that SONET path. The position of each SONET path is chosen such that each path lines up on a valid boundary based on SONET protocol constraints.

If the administrator selects the second configuration option 944c (Figs. 5d and 5e), the SONET Path wizard creates one or more valid SONET paths that fully utilize the port capacity. In the current example, where the selected port is an OC12 port, in one embodiment, the second configuration option 944c allows the administrator to quickly create four STS-3c paths (Fig. 5d) or one concatenated STS-12c (Fig. 5e). The user may select the number of paths in window 944s or the type of path in window 944t. Windows 944s and 944t are linked and, thus, always present the user with consistent options. For example, if the administrator selects 4 paths in window 944s, window 944t displays STS-3c and if the administrator selects STS-12c in window 944t, window 944s displays 1 path. Again, the SONET path wizard graphically displays the position 944d and width 944f of the SONET paths created and also displays them in SONET Path table 944g along with the default parameters assigned to each SONET path.

The third configuration option allows the administrator to custom configure a port thereby providing the administrator with more flexibility. If the administrator selects the third configuration option 944d (Fig. 5f), the SONET Path wizard displays a function window 944h. The function window provides a list of available SONET Path types 944i and also displays an allocated SONET path window 944j. In this example, only the STS-3c path type is listed in the available SONET Path types window, and if the administrator wishes to configure a single STS-12c path, then they need to select the first or second configuration option 944b or 944c. To configure one or more SONET STS-3c paths, the administrator selects the STS-3c SONET path type and then selects ADD button 944k. The SONET Path wizard adds STS-3c path 944l to the allocated SONET paths window and then displays the position 944e and width 944f of the SONET path and updates Path table 944g with a listing of that SONET path including the assigned parameters. In this example, two STS-3c paths 944l and 944m are configured in this way on the selected port. The administrator may select an allocated path (e.g., 944m or 944n) in window 944j and then select the remove button

944n to delete a configured path, or the administrator may select the clear button 944o to delete each of the configured paths from window 944j. Moreover, the administrator may select an allocated path and use up arrow 944u and down arrow 944v to change the position 944e.

In any of the SONET Path windows (Figs. 5c-5f), the administrator may select a path in the SONET path table and double click on the left mouse button or select a modify button 944p to cause the GUI to display a dialog box through which the administrator may modify the default parameters assigned to each path. The wizard validates each parameter change and prevents invalid values from being entered. The administrator may also select a cancel button 944q to exit the SONET path wizard without accepting any of the configured or modified paths. If, instead, the administrator wants to exit the SONET Path wizard and accept the configured SONET Paths, the administrator selects an OK button 944r.

Once the administrator selects the OK button, the NMS client validates the parameters as far as possible within the client's view of the device and passes (step 880, Fig. 3g) this run time / instance configuration data, including all configured SONET path parameters, to the NMS server. The NMS server validates (step 881) the data received based on its view of the world and if not correct, sends an error message to the NMS client, which notifies the administrator. Thus, the NMS server re-validates all data from the NMS clients to ensure that it is consistent with changes made by any other NMS client or by an administrator using the network device's CLI. After a successful NMS server validation, the Persistent layer software within the server uses this data to generate (step 882) SQL commands, which the server sends to the configuration database software executing on the network device. This is referred to as "persisting" the configuration change. Receipt of the SQL commands triggers a validation of the data within the network device as well. If the validation is not successful, then the network device sends an error message to the NMS server, and the NMS server sends an error message to the NMS client, which displays the error to the administrator. If the validation is successful, the configuration database software then executes (step 883) the SQL commands to fill in or change the appropriate configuration tables.

As just described, the configuration process provides a tiered approach to validation of configuration data. The NMS client validates configuration data received from an administrator according to its view of the network device. Since multiple clients may manage the same network device through the same NMS server, the NMS server re-validates received configuration data. Similarly, because the network device may be managed simultaneously by multiple NMS servers, the network device itself re-validates received configuration data. This tiered validation provides reliability and scalability to the NMS.

The configuration database software then sends (step 884) active query notices, described in more detail below, to appropriate applications executing within the network device to complete the administrator's configuration request (step 885). Active query notices may also be used to update the NMS database with the changes made to the configuration database. In addition, a Configuration Synchronization process running in the network device may also be notified through active queries when any configuration changes are made or, perhaps, only when certain configuration changes are made. As previously mentioned, the network device may be connected to multiple NMS Servers. To maintain synchronization, the Configuration Synchronization program broadcasts configuration changes to each attached NMS server. This may be accomplished by issuing reliable (i.e., over TCP) SNMP configuration change traps to each NMS server. Configuration change traps received by the NMS servers are then multicast / broadcast to all attached NMS clients. Thus, all NMS servers, NMS clients, and databases (both internal and external to the network device) remain synchronized.

Even a simple configuration request from a network administrator may require several changes to one or more configuration database tables. Under certain circumstances, all the changes may not be able to be completed. For example, the connection between the computer system executing the NMS and the network device may go down or the NMS or the network device may crash in the middle of configuring the network device. Current network management systems make configuration changes in a central data repository and pass these changes to network devices using SNMP "sets". Since changes made through SNMP are committed immediately (i.e., written to the data repository), an uncompleted configuration (series of related "sets") will

leave the network device in a partially configured state (e.g., "dangling" partial configuration records) that is different from the configuration state in the central data repository being used by the NMS. This may cause errors or a network device and/or network failure. To avoid this situation, the configuration database executes groups of SQL commands representing one configuration change as a relational database transaction, such that none of the changes are committed to the configuration database until all commands are successfully executed. The configuration database then notifies the server as to the success or failure of the configuration change and the server notifies the client. If the server receives a communication failure notification, then the server re-sends the SQL commands to re-start the configuration changes. Upon the receipt of any other type of failure, the client notifies the user.

If the administrator now selects the same port 939a (Fig. 5a), clicks the right mouse button and selects the Configure SONET Paths option in pop-up menu 943, the SONET path wizard may be displayed as shown in Fig. 5f, or alternatively, a SONET Path Configuration dialog box 945 (Fig. 5g) may be displayed. The SONET Path dialog box is similar to the SONET Path wizard except that it does not include the three configuration options 944b-944d. Similar to the SONET Path wizard, dialog box 945 displays SONET line data 945a (e.g., slot 4, port 1, OC12), SONET Path table 945g and SONET path position 945e and width 945f. The administrator may modify parameters of a configured SONET path by selecting the path in the Path table and double clicking the right mouse button or selecting a Modify button 945p. The administrator may also add a SONET path by selecting an Add button 945k, which causes the SONET path dialog box to display another SONET path in the path table. Again, the administrator may modify the parameters by selecting the new SONET path and then the Modify button. The administrator may also delete a SONET path by selecting it within the SONET Path table and then selecting a Delete button 945m. The administrator may cancel any changes made by selecting a Cancel button 945n, or the administrator may commit any changes made by selecting an OK button 945r.

The SONET path wizard provides the administrator with available and valid configuration options. The options are consistent with constraints imposed by the SONET protocol and the network device itself. The options may be further limited by other constraints, for example, customer subscription limitations. That is, ports or

modules may be associated with particular customers and the SONET Path wizard may present the administrator with configuration options that match services to which the customer is entitled and no more. For example, a particular customer may have only purchased service on two STS-3c SONET paths on an OC12 SONET port, and the SONET Path wizard may prevent the administrator from configuring more than these two STS-3c SONET paths for that customer.

By providing default values for SONET Path parameters and providing only configuration options that meet various protocol, network device and other constraints, the process of configuring SONET paths is made simpler and more efficient, the necessary expertise required to configure SONET paths is reduced and the potential for mis-configurations is reduced. In addition, as the administrator provides input to the SONET path configuration wizard, the wizard validates the input and presents the administrator with configuration options consistent with both the original constraints and the administrator's configuration choices. This further reduces the necessary expertise required to configure SONET paths and further minimizes the potential for mis-configurations. Moreover, short cuts presented to the administrator may increase the speed and efficiency of configuring SONET paths.

If the administrator now selects SONET path tab 942 (Fig. 5h), GUI 895 displays an inventory including the two STS-3c paths (942a and 942b) just configured. The SONET path tab includes information about each SONET path, such as SONET line information (e.g., shelf, slot and port), Path Position, Path Width, Ingress Connection and Egress Connection. It may also include Path Type and Service (e.g., Terminated ATM, Switched SONET), and a Path Name. The SONET Path configuration wizard may automatically assign the Path Name based on the shelf, slot and port. Parameters, such as Path Name, Path Width, Path Number and Path Type, may be changed by selecting a SONET path from the inventory and double clicking on that SONET path or selecting a Modify button (not shown) causing a dialog box to appear. The administrator may type in different parameter values or select from a pull-down list of available options within the dialog box.

Similarly, if the administrator selects an ATM Interfaces button 942c or directly selects the ATM Interfaces tab 946 (Fig 5i), GUI 895 displays an inventory including

two ATM interfaces (946a and 946b) corresponding to the two STS-3c paths just configured. The SONET Path configuration wizard automatically assigns an ATM interface name based again on the shelf, slot and port. The SONET Path wizard also automatically assigns a minimum VPI bits and maximum VPI bits and a minimum and maximum VCI bits. Again, the ATM Interfaces tab lists information such as the shelf, port and slot as well as the Path name and location of the card. The ATM Interfaces tab also lists the Virtual ATM (V-ATM) interfaces (IF) count. Since no virtual ATM interfaces have yet been configured, this value is zero and Virtual ATM Interfaces tab 947 and Virtual Connections tab 948 do not yet list any information. The administrator may return to the SONET Paths tab to configure additional SONET paths by selecting a Back button 946h or by directly selecting the SONET Paths tab.

Referring to Fig. 5j, instead of selecting a port (e.g., 939a, Fig. 5a) and then selecting a Configure SONET Paths option from a pop-up menu, the administrator may instead select a path from the inventory of paths in SONET Interfaces tab 940 and then select a Paths button 940a to cause SONET Path wizard 944 (Fig. 5k) to be displayed. For example, the administrator may select line 949a corresponding to port 941a on card 556d and then select Paths button 940a to cause SONET Path wizard 944 to be displayed. As shown, SONET line data 944a indicates that this is port two in slot 5 and is an OC48 type port. Again, the administrator is presented with three configuration options 944b, 944c and 944d.

If the administrator selects option 944b (Fig. 5l), then the SONET Path Wizard creates a single STS-48c concatenated SONET Path and inventories the new path in Path table 944g and displays the path position 944e and path width 944f. If the administrator instead selects option 944c (Figs. 5m-5o), the SONET Path wizard creates one or more valid SONET paths that fully utilize the port capacity. For example, as pull down window 944s (Fig. 5n) shows one single concatenated STS-48c path (Fig. 5n) may be created, four STS-12c paths (Fig. 5m), or sixteen STS-3c paths (Fig. 5o) may be created. Instead, the administrator may select option 944d (Fig. 5p) to custom configure the port. Again, function window 944h is displayed including a list of Available SONET Path types 944i and a list of Allocated SONET Paths 944j. In this instance where the port is an OC48, both an STS-3c and STS-12c are listed as available SONET Path types. The administrator may select one and then

select Add button 944k to add a path to the Allocated SONET Paths list and cause the wizard to display the path in Path Table 944g and to display the path position 944e and width 944f. In this example, two STS-3c paths are added in positions 1 and 4 and two STS-12c paths are added in positions 22 and 34.

Now when the administrator selects SONET Paths tab 942 (Fig. 5q), the inventory of paths includes the four new paths (942c-942f). Similarly, when the administrator selects ATM Interfaces tab 946 (Fig. 5r), the inventory of ATM interfaces includes four new interfaces (946c-946f) corresponding to the newly created SONET paths.

Instead of selecting a port in device mimic 896a and then the Configure SONET Paths option from a pop-up menu and instead of selecting a SONET interface in the SONET Interfaces tab and then selecting the Paths button, the SONET Path wizard may be accessed by the administrator from any view in the GUI by simply selecting a Wizard menu button 951 and then selecting a SONET Path option 951a (Fig. 5q) from a pull-down menu 951b. When the SONET path wizard appears, the SONET line data (i.e., slot, port and type) will be blank, and the administrator simply needs to provide this information to allow the SONET path wizard to select the appropriate port. If the administrator selects a port in the Ports tab prior to selecting the SONET path option from the wizard pull-down menu, then the SONET wizard will appear with this information displayed as the SONET line data but the administrator may modify this data to select a different port from the SONET wizard.

To create virtual connections between various ATM Interfaces / SONET Paths within the network device, the administrator first needs to create one or more virtual ATM interfaces for each ATM interface. At least two virtual ATM interfaces are required since two discrete virtual ATM interfaces are required for each virtual connection. In the case of a multipoint connection there will be one root ATM interface and many leafs. To do this, the administrator may select an ATM interface (e.g., 946b) from the inventory in the ATM Interfaces tab and then select a Virtual Interfaces button 946g to cause Virtual Interfaces tab 947 (Fig. 5s) to appear and display an inventory of all virtual interfaces associated with the selected ATM interface. In this example, no virtual ATM interfaces have yet been created, thus, none are displayed.

The Virtual ATM Interfaces tab also includes a device navigation tree 947a. The navigation tree is linked with the Virtual Interfaces button 946g (Fig. 5r) such that the device tree highlights the ATM interface (e.g., ATM-Path2_11/4, Fig. 5s) that was selected when the Virtual Interfaces button was selected. When the Virtual Interfaces button is selected, the NMS client automatically requests virtual interface data corresponding to the selected ATM interface from the NMS server and then the NMS client displays this data in the Virtual ATM Interfaces tab. This saves memory space within the NMS client since only a small amount of data relevant to the virtual ATM interfaces associated with the selected ATM interface must be stored. In addition, since the amount of data is small, the data transfer is quick and reduces network traffic.

Instead the administrator may directly select Virtual ATM Interfaces tab 947 and then use the device tree 947a to locate the ATM interface they wish to configure with one or more virtual ATM interfaces. In this instance, the NMS client may again automatically request virtual interface data from the NMS server, or instead, the NMS client may simply use data stored in cache.

To return to the ATM Interfaces tab, the administrator may select a Back button 947d or directly select the ATM Interfaces tab. Once the appropriate ATM interface has been selected (e.g., ATM-Path2_11/4/1) in the Virtual ATM Interfaces tab device tree 947a, then the administrator may select an ADD button 947b to cause a virtual ATM (V-ATM) Interfaces dialog box 950 (Fig. 5t) to appear.

GUI 895 automatically fills in dialog box 950 with default values for Connection type 950a, Version 950b and Administration Status 950c. The administrator may provide a Name or Alias 950d and may modify the other three parameters by selecting from the options provided in pull down menus. This and other dialog boxes may also have wizard-like properties. For example, only valid connection types, versions and administrative status choices are made available in corresponding pull-down menus. For instance, Version may be UNI Network 3.1, UNI Network 4.0, IISP User 3.0, IISP User 3.1, PNNI, IISP Network 3.0 or IISP Network 3.1, and Administration Status may be Up or Down. When Down is selected, the virtual ATM interface is created but not enabled. With regard to connection type, for the first virtual ATM

interface created for a particular ATM interface, the connection type choices include Direct Link or Virtual Uni. However, for any additional virtual ATM interfaces for the same ATM interface the connection type choices include only Logical Link. Hence the dialog box provides valid options to further assist the administrator. When finished, the administrator selects an OK button 950e to accept the values in the dialog box and cause the virtual ATM interface (e.g., 947c, Fig. 5u) to be inventoried in Virtual ATM tab 947.

The administrator may then select ADD button 947b again to add another virtual ATM interface to the selected ATM interface (ATM-Path2_11/4/1). Instead, the administrator may use device tree 947a to select another ATM interface, for example, ATM path 946c (Fig. 5r) designated ATM-Path1_11/5/2 (Fig. 5v) in device tree 947a. The administrator may again select the ADD button or the administrator may select port 941a on card 556d, click the right mouse button and select the "Add Virtual Connection" option from pop-up menu 943. This will again cause dialog box 950 (Fig. 5t) to appear, and the administrator may again modify parameters and then select OK button 950e to configure the virtual ATM interface.

To create a virtual connection, the administrator selects a virtual ATM interface (e.g., 947c, Fig. 5v) and then selects a Virtual Connections button 947d or a Virtual Connection option 951c (Fig. 5q) from wizard pull-down menu 951b. This causes GUI 895 to start a Virtual Connection configuration wizard 952 (Fig. 5w). Just as the SONET Path configuration wizard guides the administrator through the task of setting up a SONET Path, the Virtual Connection configuration wizard guides the administrator through the task of setting up a virtual connection. Again, the administrator is presented with valid configuration options and default parameter values are provided as a configuration starting point. As a result, the process of configuring virtual connections is simplified, and required administrator expertise is reduced since the administrator does not need to know or remember to provide each parameter value. In addition, the wizard validates configuration requests from the administrator to minimize the potential for mis-configuration.

The Virtual Connection configuration wizard includes a Connection Topology panel 952a and a Connection Type panel 952b. Within the Connection Topology panel the

administrator is asked whether they want a point-to-point or point-to-multipoint connection, and within the Connection Type panel, the administrator is asked whether they want a Virtual Path Connection (VPC) or a Virtual Channel Connection (VCC). In addition, the administrator may indicate that they want the VPC or VCC made soft (SPVPC/SPVCC). Where the administrator chooses a point-to-point, VPC connection, the Virtual Connection wizard presents dialog box 953 (Fig. 5x).

The source (e.g., test1 in End Point 1 window 953a) for the point-to-point connection is automatically set to the virtual ATM interface (e.g., 947c, Fig. 5v) selected in Virtual ATM Interface tab 947 when the virtual connection button 947d was selected. The administrator may change the source simply by selecting another virtual ATM interface in device tree 953b, for example, test2. Similarly, the administrator selects a destination (e.g., test3 in End Point 2 window 953c) for the point-to-point connection by selecting a virtual ATM interface in device tree 953d, for example, test3. If the administrator had selected point-to-multipoint in Connection Topology panel 952a (Fig. 5w), then the user would select multiple destination devices from device tree 953d or the wizard may present the administrator with multiple End Point 2 windows in which to select the multiple destination devices. In addition, if within Connection Topology panel 952b (Fig. 5w) the administrator had elected to make the VPC or VCC soft (SPVPC/SPVCC), then the user may select in End Point 2 window 953c (Fig. 5x) a virtual ATM interface in another network device.

The virtual Connection wizard also contains a Connections Parameters window 953e, an End Point 1 Parameters window 953f and an End Point 2 Parameters window 953g. Again for point-to-multipoint, there will be multiple End Point 2 Parameters windows. Within the Connections Parameters window, the administrator may provide a Connection name (e.g., test). The administrator also determines whether the connection will be configured in an Up or Down Administration Status, and may provide a Customer Name (e.g., Walmart) or select one from a customer list, which may be displayed by selecting Customer List button 953h.

Within the End Point 1 and 2 Parameters windows, the administrator provides a Virtual Path Identifier (VPI) in window 953i, 953j or selects a Use Any VPI Value indicator 953k, 953l. If the administrator chooses a VCC connection in Connection

Type window 952b (Fig. 5w), then the administrator must also provide a Virtual Channel Indicator (VCI) in window 953m, 953n or select a Use Any VCI Value indicator 953o, 953p. The administrator also selects a Transmit and a Receive Traffic Descriptor (e.g., Variable Bit Rate (VBR)-high, VBR-low, Constant Bit Rate (CBR)-high, CBR-low) from a pull down menu or selects an Add Traffic Descriptor button 953q, 953r. If the administrator selects one of the Add Traffic Descriptor buttons, then a traffic descriptor window 956 (Fig. 5y) is displayed and the administrator may add a new traffic descriptor by providing a name and selecting a quality of service (QoS) class and a traffic descriptor type from corresponding pull down menus. Depending upon the QoS class and type selected, the administrator may also be prompted to input peak cell rate (PCR), sustainable cell rate (SCR), maximum burst size (MBS) and minimum cell rate (MCR), and for each PCR, SCR, MBS and MCR, the administrator will be prompted for a cell loss priority (CLP) value where CLP=0 corresponds to high priority traffic and CLP=0+1 corresponds to combined / aggregated high and low priority traffic. The traffic descriptors indicate the priority of the traffic to be sent over the connection thereby allowing parameterization of quality of service. The administrator may select a Use the same Traffic Descriptor for both Transmit and Receive indicator 953s, 953t (Fig. 5x).

Within the Virtual Connection wizard, the administrator may select a Back button 953u (Fig. 5x) to return to screen 952 (Fig. 5w) or a Cancel button 953v to exit out of the wizard without creating a virtual connection. On the other hand, if the administrator has provided all parameters and wants to commit the virtual connection, then the administrator selects a Finish button 953w. The NMS client passes the parameters to the NMS server, which validates the data and then writes the data into the network device's configuration database. The data is validated again within the network device and then through active queries modular processes throughout the device are notified of the configuration change to cause these processes to implement the virtual connection. GUI 895 then displays the newly created virtual connection 948a (Fig. 5z) in a list within Virtual Connections tab 948. The administrator may then create multiple virtual connections between the various virtual ATM interfaces, each of which will be listed in the Virtual Connections tab 948. The administrator may also select a Back button 948b to return to the Virtual ATM Interfaces tab or select the Virtual ATM Interfaces tab directly.

The Virtual Connections tab also includes a device navigation tree 948c. The device tree is linked with Virtual Connections button 947d such that the device tree highlights the virtual ATM interface that was selected in Virtual ATM Interfaces tab 947 when the Virtual Connections button was selected. The Virtual Connections tab then only displays data relevant to the highlighted portion of the device tree.

As described above, the SONET Paths tab, ATM Interfaces tab, Virtual ATM Interfaces tab and Virtual Connections tabs are configuration tabs that are chained together providing wizard-like properties. Both the order of the tabs from right to left and the forward buttons (e.g., ATM Interfaces button 942c) and back buttons (e.g., Back button 946h) allow an administrator to easily and quickly sequence through the steps necessary to provision services. Although device navigation trees were shown in only the Virtual ATM Interface tab and the Virtual Connection tab, a device navigation tree may be included in each tab and only data relevant to the highlighted portion of the navigation tree may be displayed.

In addition to the SONET Interface and SONET Paths tabs, the status window may include tabs for other physical layer protocols, for example, Ethernet. Moreover, in addition to the ATM Interfaces and Virtual ATM Interfaces tabs, the status window may include tabs for other upper layer protocols, including MPLS, IP and Frame Relay. Importantly, other configuration wizards in addition to the SONET Path configuration wizard and Virtual Connection configuration wizard may also be used to simplify service provisioning.

Custom Navigator:

In typical network management systems, the graphical user interface (GUI) provides static choices and is not flexible. That is, the screen flow provided by the GUI is predetermined and the administrator must walk through a predetermined set of screens each time a service is to be provisioned. To provide flexibility and further simplify the steps required to provision services within a network device, GUI 895, described in detail above, may also include a custom navigator tool that facilitates "dynamic menus". When the administrator selects the custom navigator menu button 958 (Fig. 4x), a pop-up menu 958a displays a list of available "screen marks". The

list of screen marks may include default screen marks (e.g., Virtual ATM IF 958b and Virtual Connection 958c) and/or administrator created screen marks (e.g., test 958d).

When the administrator selects a particular screen mark, the custom navigator shortcuts the configuration process by jumping forward past various configuration screens to a particular configuration screen corresponding to the screen mark. For example, if the administrator selects a Virtual ATM IF screen mark 958b, the custom navigator presents the Virtual ATM Interface tab (Fig. 5u). The administrator may then select an ATM interface from device tree 947a and select Add button 947b to add a virtual ATM interface. Similarly, the administrator may select a Virtual Connection screen mark 958c, and the custom navigator automatically presents Virtual Connection wizard 952 (Fig. 5w).

Moreover, the custom navigator allows the administrator to create unique screen marks. For example, the administrator may provision SONET paths and ATM interfaces as described above, then select an ATM interface (e.g., 946b, Fig. 5r) in ATM interfaces tab 946 and select Virtual Interfaces button 946g to display Virtual ATM Interfaces tab 947 (Fig. 5s), and as described above, the devices tree 947a will highlight the selected ATM interface. If the administrator believes they may want to return to the Virtual Interfaces tab multiple times to provision multiple virtual ATM interfaces for the selected ATM interface or other ATM interfaces near the selected ATM interface in device tree 947a, then the administrator would select a screen mark button 959 to create a screen mark for this configuration position. A dialog box would appear in which the administrator enters the name of the new screen mark (e.g., test 958d, Fig. 4x) and this new screen mark name is added to the list of screen marks 958a. The custom navigator then takes a "snap shot" of the metadata necessary to recreate the screen and the current configuration position (i.e., highlight ATM-Path2_11/4/1). If the administrator now selects this screen mark while another tab is displayed, the custom navigator uses the metadata associated with the screen mark to present the screen shot displayed in Fig. 5s to the administrator updated with any other configuration changes made subsequent to the creation of the screen mark.

As a result, the administrator is provided with configuration short cuts, both default short cuts and ones created by the administrator himself. Many other screen marks

may be created through GUI 895, and in each case, the screen marks may simplify the configuration process and save the administrator configuration time.

Custom Wizard:

To provide additional flexibility and efficiency, an administrator may use a custom wizard tool to create unique custom wizards to reflect common screen sequences used by the administrator. To create a custom wizard, the administrator begins by selecting a Custom Wizard menu button 960 (Fig. 4y) to cause a pull-down menu 960a to appear and then selecting a Create Wizard 960b option from the pull-down menu. The administrator then begins using the particular sequence of screens that they wish to turn into a custom wizard and the custom wizard tool records this sequence of screens. For example, the administrator may begin by selecting a port within device mimic 896a, clicking the right mouse button and selecting the Configure SONET Paths option to cause the SONET Path configuration wizard 944 (Fig. 5b) to appear. The custom wizard tool records the first screen to be included in the new custom wizard as the SONET Path configuration wizard screen 944. After filling in the appropriate data for the current port configuration, the administrator presses the OK button and the SONET Paths tab 942 (Fig. 5h) appears. The custom wizard records the SONET Paths tab screen as the next screen in the new custom wizard. The administrator may then select Virtual ATM interfaces tab 947 (Fig. 5s) to cause this tab to be displayed. Again, the custom navigator records this screen as the next screen in the new custom wizard.

The administrator may continue to select further screens to add to the new custom wizard (for example, by selecting an ATM interface from device tree 947a and then selecting the Add button 947b to cause the Add V-ATM Interface dialog box 950 (Fig. 5t) to appear) or, if the administrator is finished sequencing through all of the screens that the administrator wants added to the new custom wizard, the administrator again selects Custom Wizard menu button 960 (Fig. 4y) and then selects a Finish Wizard option 960c. This causes a dialog box 960d to appear, and the administrator enters a name (e.g., test) for the custom wizard just created.

To access a custom wizard, the administrator again selects Custom Wizard 960 menu button and then selects a Select Wizard option 960e to cause an inventory 960f of

custom wizards to be displayed. The administrator then selects a custom wizard (e.g., test), and the custom wizard automatically presents the administrator with the first screen of that wizard. In the continuing example, the custom navigator presents SONET Path configuration wizard screen 961 (Fig. 4z). Since the administrator may start a custom wizard from any screen within GUI 895, SONET Path wizard screen 961 is different from the screen 944 displayed in Fig. 5b because SONET line data 961a (i.e., slot, port, type) is not provided. That is, the administrator may not have selected a particular SONET Path to configure prior to selecting the custom wizard. Hence, the SONET line data is blank and the administrator must fill this in. After the administrator enters and/or modifies the SONET line data and any other data within the first screen, the administrator selects a Next button 961b (or an OK button) to move to the next screen in the sequence of screens defined by the custom wizard. In the next and subsequent screens, the administrator may also select a Back button to return to a previous screen within the custom wizard screen sequence. Thus, the custom wizard tool allows an administrator to make their provisioning tasks more efficient by defining preferred screen sequences for each task.

Off-Line Configuration:

There may be times when a network manager / administrator wishes to jump-start initial configuration of a new network device before the network device is connected into the network. For example, a new network device may have been purchased and be in the process of being delivered to a particular site. Generally, a network manager will already know how they plan to use the network device to meet customer needs and, therefore, how they would like to configure the network device. Because configuring an entire network device may take considerable time once the device arrives and because the network manager may need to get the network device configured as soon as possible to meet network customer needs, many network managers would like the ability to perform preparatory configuration work prior to the network device being connected into the network.

In the current invention, network device configuration data is stored in a configuration database within the network device and all changes to the configuration database are copied in the same format to an external NMS database. Since the data in both databases (i.e., configuration and NMS) is in the same format, the present invention

allows a network device to be completely configured "off-line" by entering all configuration data into an NMS database using GUI 895 in an off-line mode. When the network device is connected to the network, the data from the NMS database is reliably downloaded to the network device as a group of SQL commands using a relational database transaction. The network device then executes the SQL commands to enter the data into the internal configuration database, and through the active query process (described below), the network device may be completely and reliably configured.

Referring to Fig. 6a, the network manager begins by selecting Devices branch 898a in navigation tree 898, clicking the right mouse button to cause pop-up menu 898c to appear and selecting the Add Devices option causing dialog box 898d (Fig. 6b) to be displayed. The network manager then enters the intended IP address or DNS name (e.g., 192.168.9.201) of the new network device into field 898e and de-selects a Manage device in on-line mode option 898k – that is, the network manager moves the cursor over box 898l and clicks the left mouse button to clears box 898l. De-selecting the Manage device in on-line mode option indicates that the network device will be configured in off-line mode. The network manager then selects Add button 898f to cause dialog box 898d to add the IP address to window 898g (Fig. 6c). However, in this example, box 898m is blank indicating the network device is to be configured off-line.

Referring to Fig. 6d, the new network device (e.g., 192.168.9.201) is now added to the list of devices 898b to be managed. However, the icon includes a visual indicator 898n (e.g., red "X") indicating the off-line status of the device. To begin off-line configuration, the network manager selects the new device. Since the NMS client and NMS server are not connected to the actual network device, no configuration data may be read from the network device's configuration database. The network manager must, therefore, populate a device mimic with modules representing the physical inventory that the network device will include. To do this, the network manager begins by clicking on the right mouse button to display pop-up menu 898o, and selects the Add Chassis option to cause a device mimic 896a (Fig. 6e) to be displayed in window 896b including only a chassis. All slots in the chassis may be empty and visually displayed, for example, in a gray or light color. Alternatively, particular

modules that are required for proper network device operation may be automatically included in the chassis. If more than one chassis type is available, a dialog box would appear and allow the network manager to select a particular chassis. In the current example, only one chassis is available and is automatically displayed when the network manager selects the Add Chassis option.

Again, the cursor provides context sensitive pop-up windows. For example, the network manager may move the cursor over a particular slot (e.g., 896c, Fig. 6e) to cause a pop-up window (e.g., 896d) to appear and describe the slot (e.g., Empty Forwarding Processor Slot Shelf 3/Slot 1). The network manager may then select an empty slot (e.g., 896c, Fig. 6f) to cause the device mimic to highlight that slot, click the right mouse button to cause a pop-up menu (e.g., 896e) to appear and select the Add Module option. In this example, only one type of forwarding card is available. Thus, it is automatically added (visually indicated in dark green, Fig. 6g) to the device mimic. This forwarding card corresponds to forwarding card 546a in Fig. 41a. The network manager may also remove a module by selecting the module (e.g., 546a), clicking the right mouse button to cause a pop-up menu 896t to appear and then selecting the Remove Module option.

If there are multiple types of modules that may be inserted in a particular slot, then a dialog box will appear after the network manager selects the Add Module option and the network manager will select the particular module that the network device will include in this slot upon delivery. For example, while viewing the back of the chassis (Fig. 6h), the manager may select an empty universal port card slot (e.g., 896f), click the right mouse button causing pop-up menu 896g (Fig. 6i) to appear and select the Add Module option. Since multiple universal port cards are available, selecting the Add Module option causes a dialog box 896h (Fig. 6j) to appear. The network manager may then select the type of universal port card to be added into the empty slot from an inventory provided in pull-down menu 896i (Fig. 6k). Once the network manager selects the appropriate card and an OK button 896j, the device mimic adds a representation of this card (e.g., 556h, Fig. 6l and see also Fig. 41b).

Typically, a network device may include many similar modules, for example, many 16 port OC3 universal port cards and many forwarding cards. Instead of having the

network manager repeat each of the steps described above to add a universal port card or a forwarding card, the network manager may simply select an inserted module (e.g., 16 port OC3 universal port card 556h, Fig. 6L) by pressing down on the left mouse button, dragging an icon to an empty slot (e.g., 556i) also requiring a similar module and releasing the left mouse button to drop a similar module (e.g., 16 port OC3 universal port card 556g, Fig. 6m) into that empty slot. Similarly, the network manager may drag and drop a forwarding card module to an empty forwarding card slot and other inserted modules into other empty slots. The network manager may use the drag and drop method to quickly populate the entire network device with the appropriate number of similar modules. To add a different type of universal port card, the network manager will again select the empty slot, click on the right mouse button, select the Add Module button from the pop-up menu and then select the appropriate type of universal port card from the dialog box.

Once the network manager is finished adding appropriate modules into the empty slots such that the device mimic represents the physical hardware that will be present in the new network device, then the network manager may configure / provision services within the network device. Off-line configuration is the same as on-line configuration, however, instead of sending the configuration data to the configuration database within the network device, the NMS server stores the configuration data in an external NMS database. After the network device arrives and the network manager connects the network device's ports into the network, the network manager selects the device (e.g., 192.168.9.201, Fig. 6n), clicks the right mouse button to cause pop-up menu 868o to appear and selects the Manage On-line option.

The NMS client notifies the NMS server that the device is now to be managed on-line. The NMS server first reconciles the physical configuration created by the network manager and stored in the NMS database against the physical configuration of the actual network device and stored in the internal configuration database. If there are any mis-matches, the NMS server notifies the NMS client, which then displays any discrepancies to the network manager. After the network manager fixes any discrepancies, the network manager may again select the Manage On-Line option in pop-up menu 898o. If there are no mis-matches between the physical device tables in the NMS database and the configuration database, then the NMS server reconciles all

service provisioning data in the NMS database against the service provisioning data in the configuration database. In this example, the network device is new and thus, the configuration database has no service provisioning data. Thus, the reconciliation will be successful.

The NMS server then instructs the network device to stop replication between the primary configuration database within the network device and the backup configuration database within the network device. The NMS server then pushes the NMS database data into the backup configuration database, and then instructs the network device to switchover from the primary configuration database to the backup configuration database. If any errors occur after the switchover, the network device may automatically switch back to the original primary configuration database. If there are no errors, then the network device is quickly and completely configured to work properly within the network while maximizing network device availability.

In the previous example, the network manager configured one new network device off-line. However, a network manager may configure many new network devices off-line. For example, a network manager may be expecting the receipt of five or more new network devices. Referring to Fig. 6o, to simplify the above process, a network manager may select an on-line device (e.g., 192.168.9.202) or off-line device (e.g., 192.168.9.201) by pressing and holding the left mouse button down, dragging an icon over to a newly added off-line device (e.g., 192.168.203) and dropping the icon over the newly added off-line device by releasing the left mouse button. The NMS client notifies the NMS server to copy the configuration data from the NMS database associated with the first network device (e.g., 192.168.9.202 or 192.168.9.201) to a new NMS database associated with the new network device and to change the data in the new NMS database to correspond to the new network device. The network manager may then select the new network device and modify any of the configuration data, as described above, to reflect the current network device requirements. As a result, off-line mode configuration is also made more efficient.

A network manager may also choose to re-configure an operational device in off-line mode without affecting the operation of the network device. For example, the network manager may want to add one or more new modules or provision services in

a network device during a time when the network sees the least amount of activity, for example, midnight. Through the off-line mode, the network manager may prepare the configuration data ahead of time.

Referring to Fig. 6p, the network manager may select an operational network device (e.g., 192.168.9.202), click on the right mouse button to cause pop-up menu 898o to appear and select the Manage On-Line option, which de-selects the current on-line mode and causes the GUI to enter an off-line mode for this device. Although the GUI has entered the off-line mode, the network device is still operating normally. The network manager may then add one or more modules and/or provision services as described above just as if the GUI were still in on-line mode, however, all configuration changes are stored by the NMS server in the NMS database corresponding to the network device instead of the network device's configuration database. Alternatively, when the NMS server is notified that a network device is to be managed off-line, the NMS server may copy the NMS database data to a temporary NMS database and store all off-line configuration changes there. When the network manager is ready (i.e., at the appropriate time and/or after adding any new modules to the network device) to download the configuration changes to the operational network device, the network manager again selects the network device (e.g., 192.168.9.202), clicks on the right mouse button to cause pop-up menu 898a to appear and selects the Manage On-Line option.

The NMS client notifies the NMS server that the device is now to be managed on-line. The NMS server first reconciles the physical configuration stored in the NMS database (or the temporary NMS database) against the physical configuration of the actual network device stored in the internal configuration database. If there are any mis-matches, the NMS server notifies the NMS client, which then displays any discrepancies to the network manager. After the network manager fixes any discrepancies, the network manager may again select the Manage On-Line option in pop-up menu 898o. If there are no mis-matches between the physical device tables in the NMS database and the configuration database, then the NMS server reconciles all service provisioning data in the NMS database (or the temporary NMS database) against the service provisioning data in the configuration database. If any conflicts are discovered, the NMS server notifies the NMS client, which displays the

discrepancies to the network manager. After fixing any discrepancies, the network manager may again select the Manage On-Line option in pop-up menu 898o.

If there are no conflicts, the NMS server instructs the network device to stop replication between the primary configuration database within the network device and the backup configuration database within the network device. The NMS server then pushes the NMS database data into the backup configuration database, and then instructs the network device to switchover from the primary configuration database to the backup configuration database. If any errors occur after the switchover, the network device may automatically switch back to the original primary configuration database. If there are no errors, then the network device is quickly re-configured to work properly within the network.

Off-line configuration, therefore, provides a powerful tool to allow network managers to prepare configuration data prior to actually implementing any configuration changes. Such preparation, allows a network manager to carefully configure a network device when they have time to consider all their options and requirements, and once the network manager is ready, the configuration changes are implemented quickly and efficiently.

FCAPS Management:

Fault, Configuration, Accounting, Performance and Security (FCAPS) management are the five functional areas of network management as defined by the International Organization for Standardization (ISO). Fault management is for detecting and resolving network faults, configuration management is for configuring and upgrading the network, accounting management is for accounting and billing for network usage, performance management is for overseeing and tuning network performance, and security management is for ensuring network security. Referring to Fig. 7a, GUI 895 provides a status button 899a-899f for each of the five FCAPS. By clicking on one of the status buttons, a status window appears and displays the status associated with the selected FCAPS button to the network administrator. For example, if the network administrator clicks on the F status button 899a, a fault event summary window 900 (Fig. 7b) appears and displays the status of any faults.

Each FCAP button may be colored according to a hierarchical color code where, for example, green means normal operation, red indicates a serious error and yellow indicates a warning status. Today there are many NMSs that indicate faults through color coded icons or other graphics. However, current NMSs do not categorize the errors or warnings into the ISO five functional areas of network management – that is, FCAPS. The color-coding and order of the FCAPS buttons provide a "status bar code" allowing a network administrator to quickly determine the category of error or warning and quickly take action to address the error or warning.

As with current NMSs, a network administrator may actively monitor the FCAPS buttons by sitting in front of the computer screen displaying the GUI. Unfortunately, network administrators do not have time to actively monitor the status of each network device —passive monitoring is required. To assist passive monitoring, the FCAPS buttons may be enlarged or "stretched" to fill a large portion of the screen, as shown in Fig. 7c. The FCAPS buttons may be stretched in a variety of ways, for example, a stretch option in a pull down menu may be selected or a mouse may be used to drag and drop the boarders of the FCAPS buttons. Stretching the FCAPS buttons allows a network administrator to view the status of each FCAP button from a distance of 40 feet or more. Once stretched, each of the five OSI management areas can be easily monitored at a distance by looking at the bar-encoded FCAPS strip. The "stretchy FCAPS" provide instant status recognition at a distance.

The network administrator may set the FCAPS buttons to represent a single network device or multiple network devices or all the network devices in a particular network. Alternatively, the network administrator may have the GUI display two or more FCAPS status bars each of which represents one or more network devices.

Although the FCAPS buttons have been described as a string of multiple stretched bars, many different types of graphics may be used to display FCAPS status. For example, different colors may be used to represent normal operation, warnings and errors, and additional colors may be added to represent particular warnings and/or errors. Instead of a bar, each letter (e.g., F) may be stretched and color-coded. Instead of a solid color, each FCAPS button may repeatedly flash or strobe a color. For example, green FCAPS buttons may remain solid (i.e., not flashing) while red

errors and yellow warnings are displayed as a flashing FCAPS button to quickly catch a network administrator's attention. As another example, green / normal operation FCAPS buttons may be a different size relative to yellow / warnings and red / errors FCAPS buttons. For example, an FCAPS button may be automatically enlarged if status changes from good operation to a warning status or an error status. In addition, the FCAPS buttons may be different sizes to allow the network administrator to distinguish between each FCAPS button from a further distance. For example, the buttons may have a graduated scale where the F button is the largest and each button is smaller down to the S button, which is the smallest. Alternatively, the F button may be the smallest while the S button is the largest, or the A button in the middle is the largest, the C and P buttons are smaller and the F and S buttons are smallest. Many variations are possible for quickly alerting a network administrator of the status of each functional area.

Referring to Fig. 7d, for more detailed FCAPS information, the network administrator may double click the left mouse button on a particular network device (e.g., 192.168.9.201) to cause device navigation tree 898 to expand and display FCAPS branches, for example, Fault branch 898p, Configuration branch 898q, Accounting branch 898r, Performance branch 898s and Security branch 898t. The administrator may then select one of these branches to cause status window 897 to display tabs / folders of data corresponding to the selected branch. For example, if Fault branch 898p is selected (Fig. 7e), an Events tab 957a is displayed in status window 897 as well as tab holders for other tabs (e.g., System Log tab 957b (Fig. 7f) and Trap Destinations 957c (Fig. 7g)). If the administrator double clicks the left mouse button on the Fault branch, then device tree 898 displays a list 958a of the available fault tabs. The administrator may then select a tab by selecting the tab holder from status window 897 or device tree 898.

Events tab 957a (Fig. 7e) displays an event number, date, time, source, category and description of each fault associated with a module or port selected in device mimic 896a. System Log tab 957b (Fig. 7f) displays an event number, date, time, source, category and description of each fault associated with the entire network device (e.g., 192.168.9.201), and Trap Destination tab 957c (Fig. 7g) displays a system / network device IP address or DNS name, port and status corresponding to each detected trap

destination. Various other tabs and formats for displaying fault information may also be provided.

Referring to Fig. 7h, if the administrator double clicks the left mouse button on Configuration branch 898q, then device tree 898 expands to display a list 958b of available configuration sub-branches, for example, ATM protocol sub-branch 958c, System sub-branch 958d and Virtual Connections sub-branch 958e. When the device branch (e.g., 192.168.9.201), Configuration branch 898q or System branch 958d is selected, System tab 934, Module tab 936, Ports tab 938, SONET Interface tab 940, SONET Paths tab 942, ATM Interfaces tab 946, Virtual ATM Interfaces tab 947 and Virtual Connections tab 948 are displayed. These configuration tabs are described above in detail (see Figs. 4s-4z and 5a-5z).

If ATM protocol branch 958c is selected, then tabs / folders holding ATM protocol information are displayed, for example, Private Network-to-Network Interface (PNNI) tab 959 (Fig. 7i). The PNNI tab may display PNNI cache information such as maximum path (per node), maximum entries (nodes), timer frequency (seconds), age out (seconds) and recently referenced (seconds) data. The PNNI tab may also display PNNI node information for each PNNI node such as domain name, administrative status, ATM address and node level. The PNNI cache and PNNI node information may be for a particular ATM interface, all ATM interfaces in the network device or ATM interfaces corresponding to a port or module selected by the administrator in device mimic 896a. Various other tabs displaying ATM information, for example, an Interim Link Management Interface (ILMI) tab, may also be provided. In addition, various other upper layer network protocol branches may be included in list 958b, for example, MuliProtocol Label Switching (MPLS) protocol, Frame Relay protocol or Internet Protocol (IP) branches, depending upon the capabilities of the selected network device. Moreover, various physical layer network protocol branches (and corresponding tabs) may also be included, for example, Synchronous Optical NETwork (SONET) protocol and/or Ethernet protocol branches, depending upon the capabilities of the selected network device.

If Virtual Connections branch 958e is selected, then tabs / folders holding virtual connection information are displayed, for example, Soft Permanent Virtual Circuit

(PVC) tab 960a (Fig. 7j) and Switched Virtual Circuits tab 960b (Fig. 7k). Soft PVC tab 960a may display information relating to source interface, Virtual Path Identifier (VPI), Virtual Channel Identifier (VCI), status, date and time. Switched Virtual Circuits tab 960b may display information relating to interface, VPI, VCI, address format, address, status, date and time. The information in either tab may be for a particular virtual connection, all virtual connections in the network device or only those virtual connections corresponding to a port or module selected by the administrator in device mimic 896a. Various other tabs displaying virtual connection information, for example, virtual connections established through various different upper layer network protocols, may also be provided, depending upon the capabilities of the selected network device.

For detailed accounting information, the administrator may select Accounting branch 898r (Fig. 7l). This will cause one or more tabs / folders to be displayed which contain accounting data. For example, a Collection Setup tab 961 may be displayed that provides details on a primary and a backup archive host – that is, the system executing the Data Collection Server (described above). The Collection Setup tab may also provide statistics timer data and backup file storage data. Various other tabs displaying accounting information may also be provided. For example, a tab may be created for each particular customer to track the details of each account.

For detailed performance information, the administrator may select Performance branch 898s (Fig. 7m) and double click the left mouse button to review a list 958f of available sub-branches, for example, ATM sub-branch 958g, Connections sub-branch 958h, Interfaces sub-branch 958i, System sub-branch 958j, and SONET sub-branch 958k. Selecting Performance branch 898s or System sub-branch 958j provides general performance tabs in stats window 897, for example, System tab 962a and Fans tab 962b (Fig. 7n). System tab 962a may provide graphical representations of various system performance parameters, for example, an odometer style graphic may be used to display CPU Utilization 962c and power supply voltage level 962e and 962f and a temperature gauge may be used to show Chassis Temperature 962d. Fans tab 962b may provide graphical representations of the status of the network device's fans. For example, fans may be colored green and shown spinning for normal operation, yellow and spinning for a warning status and red and not spinning for a

failure status. Various other graphical representations may be used, for example, bar graphs or pie charts, and instead of graphical representations, the data may be provided in a table or other type of format. Moreover, the data in the other tabs displayed in status window 897 may also be displayed in various formats including graphical representations.

If the administrator selects ATM sub-branch 958g (Fig. 7o), various tabs are displayed containing ATM related performance information, for example, ATM Stats In tab 963a, ATM Stats out tab 963b (Fig. 7p), Operations Administration Maintenance (OAM) Performance tab 963c (Fig. 7q), OAM Loopback tab 963d (Fig. 7r), ATM Switched Virtual Circuit (SVC) In tab 963e (Fig. 7s), ATM SVC Out tab 963f (Fig. 7t), ATM Signaling ATM Adaptation Layer (SAAL) In tab 963g (Fig. 7u) and ATM SAAL Out tab 963h (Fig. 7v). The data displayed in each of these tabs may correspond to a particular ATM path (e.g., ATM-Path1_11/2/1), to all ATM paths corresponding to a particular port or module selected by the administrator in device mimic 896a or to all the ATM paths in the network device. ATM Stats In tab 963a (Fig. 7o) and ATM Stats Out tab 963b (Fig. 7p) may display, for example, the type, description, cells, cells per second and bits per second for each ATM path. OAM Performance tab 963c (Fig. 7q) may display, for example, VPI, VCI, status, session type, sink source, block size and end point statistics for each ATM path, while OAM Loopback tab 963d (Fig. 7r) may display, for example, VPI, VCI, status, send count, send trap, endpoint and flow statistics for each ATM path. ATM SVC In tab 963e (Fig. 7s) and ATM SVC Out tab 963f (Fig. 7t) may display, for example, type, description, total, connected, failures, last cause and setup Protocol Data Unit (PDU) data for each path, and ATM SAAL In tab 963g (Fig. 7u) and ATM SAAL Out tab 963h (Fig. 7v) may display, for example, type, description, errors, discards, begin PDUs, begin acknowledge, PDU begin and End PDUs for each ATM path. Various other upper layer network protocol sub-branches may also be displayed in list 958f, including a sub-branch for MPLS, Frame Relay and/or IP, depending upon the capabilities of the selected network device.

If the administrator selects Connections sub-branch 958h (Fig. 7w), various tabs are displayed containing connection related performance information, for example, ATM Connection tab 964a and Priority tab 964b (Fig. 7x). ATM Connection tab 964a may

include, for example, connection name, transmit, receive cell loss ratio, cell discard total and throughput data for particular ATM connections. Priority tab 964b may include, for example, connection name, Cell Loss Priority (CLP) 0 transmit, CLP1 receive, transmit total, CLP0 receive, CLP1 receive and receive total data for particular ATM connections. The data in either tab may be for a particular selected ATM connection, each ATM connection in the network device or only those ATM connections corresponding to a particular port or module selected by the administrator in device mimic 896a.

If the administrator selects Interfaces sub-branch 958i (Fig. 7y), various tabs are displayed containing interface related performance information, for example, Interfaces tab 965. Interfaces tab 965 may include, for example, slot and port location, description, type, speed, in octets, out octets, in errors, out errors, in discards and out discards data for particular ATM interfaces. The data in the tab may be for a particular selected ATM interface, each ATM interface in the network device or only those ATM interfaces corresponding to a particular port or module selected by the administrator in device mimic 896a.

Referring to Fig. 8a, if the administrator selects SONET sub-branch 958k, various tabs are displayed containing SONET related performance information, for example, Section tab 966a, Line tab 966b (Fig. 8b) and Synchronous Transport Signal (STS) Path tab 966c (Fig. 8c). Each of the three tabs displays a shelf/slot/port location, port descriptor, status, errored seconds, severely errored seconds and coding violation data for each port. The data may correspond to a particular port selected by the administrator, all ports in a selected module or all ports in the entire network device. Various other physical layer network protocol sub-branches may also be displayed in list 958f, including a sub-branch for Ethernet, depending upon the capabilities of the selected network device.

Referring to Fig. 8d, if the administrator selects Security branch 898t, various tabs are displayed containing security related information, for example, Simple Network Management Protocol (SNMP) tab 967a and Configuration Changes tab 967b (Fig. 8e). SNMP tab 967a may display, for example, read and read/write community strings and a command line interpreter (CLI) administrator password for the network

device. Configuration Changes tab 967b may display configuration changes made to the network device including event, time, configurer and workstation identification from where the change was made. Various other security tabs may also be provided.

Dynamic Bulletin Boards:

Graphical User Interface (GUI) 895 described in detail above provides a great deal of information to a network administrator to assist the administrator in managing each network device in a telecommunications network. As shown, however, this information is contained in a large number of GUI screens / tabs. There may be many instances when a network administrator may want to simultaneously view multiple screens / tabs. To provide network managers with more control and flexibility personal application bulletin boards (PABBs, i.e., dynamic bulletin boards) are provided that allow the network administrator to customize the information they view by dragging and dropping various GUI screens / tabs (e.g., windows, table entries, dialog boxes, panels, device mimics, etc.) from GUI 895 onto one or more dynamic bulletin boards. This allows the administrator to consolidate several GUI screens and/or dialog boxes into a single view. The information in the dynamic bulletin board remains linked to the GUI such that both the GUI and the bulletin boards are dynamically updated if the screens in either the GUI or in the bulletin boards are changed. As a result, the administrator may manage and/or configure network devices through the GUI screens or the dynamic bulletin board. Within the dynamic bulletin boards, the administrator may change the format of the data and, perhaps, view the same data in multiple formats simultaneously. Moreover, the administrator may add information to one dynamic bulletin board from multiple different network devices to allow the administrator to simultaneously manage and/or configure the multiple network devices. The dynamic bulletin boards provide an alternative viewing environment, and administrators can, therefore, choose what they want to view, when they want to view it and how they want to view it.

Referring to Fig. 9a, to open a dynamic bulletin board, a network administrator selects a Bulletin Bd option 968a from a view pull-down menu 968b. A bulletin board 970a (Fig. 9b) is then displayed for the administrator. Instead, a bulletin board may automatically be opened whenever an administrator logs into an NMS client to access GUI 895. Once the bulletin board is opened, the administrator may use a mouse to

move a cursor over a desired GUI screen, press and hold down a left mouse button and drag the selected item onto the bulletin board (i.e., "drag and drop"). If an item within a GUI screen is capable of being dragged and dropped (i.e., posted) to the bulletin board – that is, the bulletin board supports / recognizes the GUI object --, a drag and drop icon appears as the administrator drags the cursor over to the bulletin board. If no icon appears, then the selected item is not supported by the bulletin board. Thus, the administrator is provided with visual feedback as to whether or not an item is supported by the PABB.

Referring to Fig. 9b, as one example, an administrator may select ATM Stats In tab 963a corresponding to a particular network device (e.g., system 192.168.9.201) and drag and drop (indicated by arrow 969a) that tab onto bulletin board 970a. Since this is the first item dropped into the bulletin board, the ATM Stats In tab is sized and positioned to use the entire space (or a large portion of the space) dedicated to the bulletin board. Instead of selecting the entire ATM Stats In tab, the administrator may drag and drop only one or only a few entries from the tab, for example, entry 963i, and only those entries would then be displayed in the bulletin board. An item in bulletin board 970a may be removed by clicking on delete button 971a. The size of the bulletin board may be increased or decreased by clicking on expand button 971b or by selecting, dragging and dropping a bulletin board boarder (e.g., 971c-971f), and the bulletin board may be minimized by clicking on minimize button 971g.

The administrator may then select other GUI data to drag and drop onto bulletin board 970a. Referring to Fig. 9c, for example, the administrator may select ATM Stats Out tab 963b also corresponding to the same network device and drag and drop (indicated by arrow 969b) that tab onto bulletin board 970a. The bulletin board automatically splits the screen to include both the ATM Stats In tab 963a and the ATM Stats Out tab 963b. Now the administrator may view both of these screens simultaneously, and since the bulletin board and the screens it displays are linked to GUI 895, the ATM Stats In and Out tabs are automatically updated with information as the GUI itself is updated with information. Thus, if the administrator changes any data in the items dragged to the bulletin board, the GUI is automatically updated and if any data in the GUI is changed, then any corresponding screens in the bulletin board are also updated. Again, instead of selecting the entire tab, the administrator may select one

or more entries in a tab and drag and drop those entries onto the bulletin board. Also, the administrator may delete any bulletin board entry by clicking on the corresponding delete button 971a, and change the size of any bulletin board entry using expand button 971b or minimize button 971g.

The administrator may then select other GUI data from the same network device (e.g., system 192.168.9.201) to drag and drop to the bulletin board or the administrator may select a different network device (e.g., system 192.168.9.202, Fig. 9d) in navigation tree 898 and drag and drop various GUI screens corresponding to that network device to bulletin board 970a. For example, the administrator may select ATM Stats In tab 972a and drag and drop (indicated by arrow 969c) that tab to bulletin board 970a, and the administrator may then select ATM Stats Out tab 972b (Fig. 9e) corresponding to system 192.168.9.202 and drag and drop (indicated by arrow 969d) that tab onto bulletin board 970a. Consequently, the administrator is able to simultaneously view multiple screens corresponding to different network devices. The administrator may also choose to drag and drop related screens. For example, ATM Stats In and Out tabs 963a, 972a and 963b, 972b, respectively, may represent two ends of an ATM connection between the two network devices, and viewing these screens simultaneously may assist the administrator in managing both network devices.

As shown in Figs. 9b-9e, when new items are dropped onto the bulletin board, the bulletin board continues to divide the available space to fit the new items and may shrink the items to fit in the available space. Many more items may be added to a bulletin board, for example eight to ten items. However, instead of continuing to add items to the same bulletin board, the administrator may choose to open multiple bulletin boards (e.g., 970a-970n, Fig. 9f).

An administrator may wish to view an item dragged to a bulletin board in a different format than that displayed in the GUI. The different format may, for example, have more meaning to them or provide more clarity to the task at hand. For instance, after dragging and dropping ATM Stats In tab 963a to bulletin board 970a (Fig. 9g), the administrator may then move the cursor over the ATM Stats In tab and double click the right mouse button to cause a pull-down menu 973 displaying various format options to appear. A normal format option 973a may cause the item to appear as it

did in the GUI – that is, ATM Stats In tab 963a will appear as shown in Fig. 9g. A list format option 973b may cause the data in ATM Stats In tab 963a to be displayed as an ordered list 974a as shown in Fig. 9h. A graph option 973c may cause the data in ATM Stats In tab 963a to be displayed as a pie chart 974b (Fig. 9i), a bar graph 974c (Fig. 9j) or any other type of graph or graphical representation. A config option 973d may cause the data in the ATM Stats In tab 963a to be displayed as a dialog box 974d (Fig. 9k) displaying configuration data corresponding to a selected one of the ATM paths within the ATM Stats In tab. The data in a bulletin board entry may be displayed in a variety of different ways to make the administrator's tasks simpler and more efficient.

Referring to Fig. 9l, an administrator may wish to view an item dragged to a bulletin board in multiple different formats simultaneously. For example, the administrator may move the cursor over ATM Stats In tab 963a in the bulletin board, press down and hold the left mouse button and drag the cursor (indicated by arrow 969e) over a blank area of the bulletin board (i.e., drag and drop) to add a second copy of ATM Stats In tab 963a to the bulletin board. The administrator may then move the cursor over the copied ATM Stats In tab, double click the right mouse button to cause pull-down menu 973 to appear and select a different format in which to display the copied ATM Stats In tab. As a result, the administrator is able to simultaneously view the normal format while also viewing another format, for example, a pie chart.

Although the above examples used the ATM Stats In and Out tabs, it is to be understood that any of the tabs or entries within tabs in status window 897 may be capable of being dragged and dropped into one or more dynamic bulletin boards. In addition, an administrator may drag and drop one or more of the FCAPS buttons 899a-899e (Fig. 7a) to a bulletin board.

Referring to Fig. 9m, in addition to dragging and dropping items from status window 897 or the FCAPS buttons, an administrator may drag and drop (indicated by arrow 969f) device mimic 896a onto bulletin board 970a. In this example, the administrator has dragged and dropped the device mimic corresponding to network device 192.168.9.201. As previously mentioned, the device mimic may display ports and modules in different colors to indicate status for those components, for example,

green for normal operation, yellow for warning status and red for failure status. The administrator may then monitor the device mimic in the bulletin board while continuing to use GUI 895 for other configuration and management operations. Instead, the administrator may only select, drag and drop portions of the device mimic, for example, only one or more universal port cards or one or more forwarding cards.

Referring to Fig. 9n, the administrator may also select a different network device in navigation tree 898 and then drag and drop (indicated by arrow 969g) a device mimic 975 corresponding to that device onto bulletin board 970a. As a result, the administrator may simultaneously view the device mimics of both network devices (or more than two network devices). In addition, the administrator may drag and drop both a front and a back view of a device mimic such that all of a network device's modules may be visible. Instead, the administrator may drag and drop a front and back view 955a, 955b (Fig. 4n) from a separate pull away window 955.

A network administrator may save one or more dynamic bulletin boards before exiting out of the NMS client, and the NMS client may persist this data in the administrator's profile (described below). When the administrator logs in to the same or a different NMS client and selects Bulletin Bd option 968a (Fig. 9a), their profile may automatically open up any saved dynamic bulletin boards or present the administrator with a list of saved dynamic bulletin boards that the administrator may select to have opened. When saved dynamic bulletin boards are re-opened, the NMS client updates any items posted in those bulletin boards such that the posted items are synchronized with the GUI. Instead, the NMS client may automatically open any saved dynamic bulletin boards as soon as the administrator logs on – that is, without requiring the administrator to select Bulletin Bd option 968.

Through saved bulletin boards, a senior administrator may guide and instruct junior administrators through various tasks. For example, a senior administrator may drag and drop a sequence of GUI screens onto one or more bulletin boards where the sequence of GUI screens represent a series of steps that the senior administrator wants the junior administrator to take to complete a particular task (e.g., provisioning a SONET path). In addition to providing the series of steps, the senior administrator

may fill in various parameters (e.g., traffic descriptors) to indicate to junior administrators the default parameters the senior administrator wants them to use. The saved bulletin board may then be added to the junior administrator's profile or put in a master profile accessible by multiple users. The junior administrator may then use a saved bulletin board to interactively complete provisioning tasks similar to the task shown in the saved bulletin board. For example, the junior administrator may use the saved SONET path bulletin board to provision one or more different SONET paths. In effect, then saved bulletin boards behave as custom wizards.

As described above, the dynamic bulletin boards allow a network administrator to actively monitor – simultaneously -- specific information about one or more operational network devices. This provides a powerful customization tool for the administrator of large, complex network devices in large, complex telecommunications networks. By customizing views of one or more devices, the administrator may view only the data they need to see and in a format that best meets their needs.

Custom Object Collections:

As described above with respect to FCAPS management, a network device (e.g., 10, Fig. 1 and 540, Fig. 35) may include a large number (e.g., millions) of configurable / manageable objects such as modules, ports, paths, connections, etc. To provide flexibility and scalability, the network management system (NMS) allows users to create custom object collections. Thus, even though a network device or multiple network devices in a telecommunication network may include millions of objects, a network manager may create a collection and add only objects of interest to that collection. The objects may be of a similar or different type and may correspond to the same or different network devices. The network manager may also add and remove objects from existing collections, create additional new collections and remove existing collections. The network manager may then view the various objects in each collection. In addition, the collections are linked to the NMS graphical user interface (GUI), such that changes to objects in either are updated in the other. Custom object collections provide scalability and flexibility. In addition, custom object collections may be tied to user profiles to limit access. For example, a customer may be limited to viewing only the collections of objects related to their

account. Similarly, a network manager may be limited to viewing only those collections of objects for which they have authority.

Referring to Fig. 10a, when a user first logs into an NMS client by supplying a username and password, a list of network devices (e.g., 192.168.9.201 and 192.168.9.202) is displayed in accordance with the user's profile. Profiles are described in more detail below. In addition, a list of collections that correspond with the user's profile may also be provided. For example, navigation tree 898 may include a network branch 976a, and if the user double clicks the left mouse button on the network branch a Collections branch 976b is displayed. Similarly, if the user double clicks the left mouse button on the Collections branch, a list 976c is provided of available collections (e.g., Test1, New1, Walmart, Kmart). Alternatively or in addition, the user may select a Collections option 977a from a view pull-down menu 977b to display list 976c of available collections. List 976c may include collections pre-defined by other users (e.g., senior network administrator) and/or custom collections previously created by the user.

Referring to Fig. 10b, to view collections that include objects corresponding to only one network device, the user may select a network device (e.g., 192.168.9.201) and select a Collections option 958m. If the user double clicks the left mouse button on Collections option 958m, a list 958n (e.g., Test1 and New1) of available collections corresponding to the selected network device is displayed. In addition, as the user selects various FCAPS tabs, collections containing objects from the selected tab may be displayed. For example, collection Test1 (Fig. 10c) in navigation tree 947a may include objects selected from Virtual ATM Interfaces tab 947 and is therefore displayed when the Virtual ATM Interfaces tab is selected.

Referring to Fig. 10d, to add an object to an existing or new collection, a network manager first selects the object (e.g., Module object 978a) and then selects a Collection button 979a to cause an Add to Collection option 979b and a New Collection option 979c to appear. If the network manager selects New Collection option 979c, then a dialog box 979d (Fig. 10e) appears and the network manager inputs the name of the new collection. After inputting the name of the new collection, the network manager selects OK button 979e and the object is automatically added to

the collection and dialog box 979d is closed. If the network manager selects Add to Collection option 979b, a dialog box 979f (Fig. 10f) appears listing the available collections. The user may then select one of the listed collections and then select OK button 979g to add the object to the collection and close dialog box 979f.

Alternatively, the network manager may add an object to a collection by dragging and dropping an object from an FCAPs tab onto a collection branch in a navigation tree. Referring to Fig. 10g, for example, a network manager may select an object 978b by pressing down on the left mouse button, dragging (indicated by arrows 980a and 980b) the object to a collection and dropping the object on the collection (i.e., drag and drop). For instance, object 978b may be dragged and dropped on collection Test1 in either navigation tree 947a or 898. An object may also be dragged and dropped into a named collection in a pull down menu or dialog box.

When a collection is selected by a network manager, customer or other user, for example, by double clicking on the collection name in a navigation tree or pull down menu, the tabs in service status window 897 are changed to include only objects in the selected collection. For instance, if the collection includes only SONET path objects, then only the SONET Paths tab will include objects once the collection is selected and all other tabs will not include any objects. Alternatively, the other tabs in service status window 897 may include objects corresponding to or related to the objects in the selected collection.

Referring to Fig. 10h, when device 192.168.9.201 is selected and the SONET Paths tab is selected, a large number of SONET paths may be displayed. Referring to Fig. 10i, when collection New1 is selected, the SONET Paths Tab is changed to display only those SONET path objects within the New1 collection. As a result, the user need only view the objects in which they are interested.

To remove an object from a collection, the network manager selects an object and then selects a Remove button 982. The network manager may also select an object and double click the left mouse button to cause a dialog box to appear. The network manager may edit certain parameters and then exit from the dialog box. Any changes made to an object in a collection are automatically updated in GUI 895. Similarly,

any changes made to an object in GUI 895 are automatically updated in any and all collections including that object.

Custom object collections allow a user to view only those objects that are of interest. These may be a few objects from an otherwise very large object list in the same FCAPS tab (that is, the collection acts as a filter), and these may be a few objects from different FCAPS tabs (that is, the collection acts as an aggregator). Consequently, both flexibility and scalability are provided through custom object collections.

Custom object collections may also be used to restrict access to network objects. For example, a senior network administrator may establish a collection of objects and provide access to that collection to a junior network manager through the junior network manager's profile. In one embodiment, the junior network manager may not be provided with the full navigation tree 898 (Fig. 10a) after logging in. Instead, only a list of available collections may be provided. Thus, the junior network manager's access to the network is limited to the objects contained in the available collections and the FCAPS tabs will similarly only include those same objects.

Similarly, collections may be created that include objects corresponding to a particular customer, for example, Walmart or Kmart. A customer profile may be established for each customer and one or more collections containing only objects relevant to each customer may be assigned to the relevant customer profile. Consequently, each customer is limited to viewing only those objects corresponding to their own accounts and not the accounts of any other customers. This permits Customer Network Management (CNM) without breaching the security provided to each customer account.

Profiles:

Profiles may be used by the NMS client to provide individual users (e.g., network managers and customers) with customized graphical user interfaces (GUIs) or views of their network and with defined management capabilities. For example, some network managers are only responsible for a certain set of devices in the network. Displaying all network devices makes their management tasks more difficult and may

inadvertently provide them with management capabilities over network devices for which they are not responsible or authorized to perform. With respect to customers, profiles limit access to only those network device resources in a particular customer's network – that is, only those network device resources for which the customer has subscribed / paid. This is crucial to protecting the proprietary nature of each customer's network. Profiles also allow each network manager and customer to customize the GUI into a presentation format that is most efficient or easy for them to use. For example, even two users with access to the same network devices and having the same management capabilities may have different GUI customizations through their profiles. In addition, profiles may be used to provide other important information, for example, SNMP community strings to allow an NMS server to communicate with a network device over SNMP, SNMP retry and timeout values, and which NMS servers to use, for example, primary and secondary servers may be identified.

A network administrator is typically someone who powers up a network device for the first time, installs necessary software on the new network device as well as installs any NMS software on an NMS computer system, and adds any additional hardware and/or software to a network device. The network administrator is also the person that attaches physical network cables to network device ports. The first time GUI 895 is displayed to a network administrator, an NMS client application uses a default profile including a set of default values. Referring again to Fig. 7a, the administrator may change the default values in his profile by selecting (e.g., clicking on) a profile selection 902 in a navigation tree / menu 898. This causes the NMS client to display a profiles tab 903 (Fig. 11a) on the screen. The profile tab displays any existing profiles 904. The first time the profile tab appears only the network administrator's profile is displayed as no other profiles yet exist.

To save a network manager's time, the profiles tab may also include a copy button 906. By selecting a profile 904 and clicking on the copy button, an existing profile is copied. The network manager may then change the parameters within the copied profile. This is helpful where two user profiles are to include the same or similar parameters.

To change the parameters in the network administrator's profile or any other existing profile, including a copied profile, the user double clicks on one of the profiles 904. To add a new profile, the user clicks on an Add button 905. In either case, the NMS client displays a profile dialog box 907 (Fig. 11b) on the screen. Through the profile dialog box, a user's user name 908a, password 908b and confirmed password 908c may be added or changed. The confirm password field is used to assure that the password was entered properly in the password field. The password and confirmed password may be encrypted strings used for user authentication. These fields will be displayed as asterisks on the screen. Once added, a user simply logs on to an NMS client with this user name and password and the NMS client displays the GUI in accordance with the other parameters of this profile.

A group level access field 908d enables/disables various management capabilities (i.e., functionality available through the NMS client). Clicking on the group level access field may provide a list of available access levels. In one embodiment, access levels may include administrator, provisioner and viewer (e.g., customer), with administrator having the highest level of management capabilities and viewer having the lowest level of management capabilities (described in more detail below). In one embodiment, users can create profiles for other users at or below their own group access level. For example, a user at the provisioner access level can create user profiles for users at either the provisioner or viewer level but cannot create an administrator user profile.

A description may be added in a description field 908e, including, for example, a description of the user, phone number, fax number and/or e-mail address. A group name may be added to group field 908f, and a list of network device IP addresses may be provided in a device list field 908g. Alternatively, a domain name server (DNS) name may be provided and a host look up may be used to access the IP address of the corresponding device. Where a group name is provided, the list of network devices is associated with the group such that if the same group name is assigned to multiple user profiles, the users will be presented with the same view – that is, the same list of network devices in device list field 908g. For example, users from the same customer may share a group name corresponding to that customer. A wildcard feature is

available for the group field. For example, perhaps an * or ALL may be used as a wildcard to indicate that a particular user is authorized to see all network devices. In most instances, the wildcard feature will only be used for a high-level network administrator. The list of devices indicates which network devices the user may manage or view, for example, configuration status and statistics data may be viewed.

Within a profile certain policy flags (i.e., attributes) may also be set. For example, a flag 908h may be set to indicate that the user is not allowed to change his/her password, and an account disable flag 908i may be set to disable a particular profile / account. In addition, a flag 908j may be set to allow the user to add network device IP addresses to device list field 908g, and a number may be added to a timeout field 908k to specify a number of minutes after which a user will be automatically logged out due to inactivity. A zero in this field or no value in this field may be used to indicate unlimited activity, that is, the user will never be automatically logged out.

The profile may also be used to indicate with which NMS servers the NMS client should communicate. An IP address or DNS name may be added to a primary server field 908l and a secondary server field 908m. If the primary server fails, the client will access the secondary server. A port number may be added to primary server port field 908n and to secondary server port field 908o to indicate the particular ports that should be used for RMI connectivity to the primary and secondary NMS servers.

As described below, the information provided in a user profile is stored in tables within the NMS database, and when a user logs onto the network through an NMS client, the NMS client connects to an NMS server that retrieves the user's profile information and sends the information to the NMS client. The NMS client automatically saves the NMS server primary and secondary IP addresses and port numbers from the user's profile to a team session file associated with the user's username and password in a memory 986 (Fig 11w) local to the NMS client. If the user logs into an NMS client through a web browser, then the NMS client may save the NMS server primary and secondary IP addresses and port numbers to a cookie that is then stored in the user's local hard drive. The next time the user logs in to the NMS

client, the NMS client uses the IP addresses and port numbers stored in the team session file or cookie to connect to the appropriate NMS server.

The first time a user accesses an NMS client, however, no team session file or cookie will be available. Consequently, during the initial access of the NMS client, the NMS client may use a default IP address to connect with an NMS server or a pop-up menu 1034 (Fig. 11x) may be displayed in which the user may type in the IP address in a field 1034a of the NMS server they want the NMS client to use or select an IP address from a pop-up menu that appears when a dropdown button 1034b is selected.

User profiles and team session files / cookies allow a network administrator or provisioner to push down new NMS server IP addresses, port numbers and other information to users simply by changing those values in the user profiles. For example, an NMS server may be over loaded and a network administrator may wish to move some users from this NMS server to another less utilized NMS server. The administrator need only change the NMS server IP addresses and port numbers in the users' profiles to affect the switch. The NMS server sends the new IP addresses and port numbers to the one or more NMS clients through which the users are logged in, and the NMS clients save the new IP addresses and port numbers in each user's team session file or cookie. The next time the users log in, the NMS client(s) use the new IP addresses and port numbers in the team session files or cookies to access the appropriate NMS server. Thus, the users selected by the administrator are automatically moved to a different NMS server without the need to notify those users or take additional steps. In addition to saving IP addresses and perhaps port numbers in team session files / cookies, other information from the user profile may also be saved in team session files / cookies and changes to that information may be pushed down by the administrator simply by changing a user profile.

Referring again to Fig. 11b, additional fields may be added to device list 908g to provide more information. For example, a read field 908p may be used to indicate the SNMP community string to be used to allow the NMS server to communicate with the network device over SNMP. The SNMP connection may be used to retrieve

statistical data and device status from the network device. In addition, a read/write field 908q may be used to indicate an SNMP community string to allow the NMS server to configure the network device and/or provision services. The profile may also include a retry field 908r and a timeout field 908s to provide SNMP retry and timeout values. Many different fields may be provided in a profile.

Instead of providing all the parameters and fields in a single profile dialog box, they may be separated into a variety of a tabbed dialog boxes (Figs. 11c-11f). The tabbed dialog boxes may provide better scalability and flexibility for future needs.

In one embodiment, an administrator level user has both read and write access to the physical and logical objects of the NMS client. Thus, all screens and functionality are available to an administrator level user, and an administrator after physically attaching an external network attachment to a particular network device port may then enable that port and provision SONET paths on that port. All screens are available to a provisioner level user, however, they do not have access to all functionality as they are limited to read-only access of physical objects. For example, a provisioner can see SONET ports available on a device and can provision SONET paths on a port, but the provisioner cannot enable/disable a SONET port. In other words, a provisioner's power begins at the start of logical objects (not physical objects), for example, SONET paths, ATM interfaces, virtual ATM interfaces, and PVCs, and continues through all the configuration aspects of any object or entity that can be stacked on top of either a SONET path or ATM interface. A viewer (e.g., customer) level user has read-only access to logical entities and only those logical entities corresponding to their group name or listed in the device list field. A viewer may or may not have access to Fault, Configuration, Accounting, and Security categories of FCAPS relative to their devices.

A customer may install an NMS client at a customer site or, preferably, the customer will use a web browser to access the NMS client. To use the web browser, a service provider gives the customer an IP address corresponding to the service provider's site. The customer supplies the IP address to their web browser and while at the service

provider site, the customer logs in with their username and password. The NMS client then displays the customer level GUI corresponding to that username and password.

Referring to Fig. 11g, a user preference dialog box 909 may be used to customize the GUI into a presentation format that is most efficient or easy for a user to work with. For example, show flags (i.e., attributes) may be used to add tool tips (flag 910a), add horizontal grid lines on tables (flag 910b), add vertical grid lines on tables (flag 910c) and add bookmarks / short cuts (e.g., create a short cut to a PVC dialog box). Look and feel flags may also be used to make the GUI appear as a JAVA GUI would appear (flag 911a) or as a native application, for example, Windows, Windows/NT or Motif, GUI would appear (flag 911b).

As an alternative to providing a Group Name 908f (Fig. 11b) or a Customer Name (Fig. 11c), when a profile is created or changed the administrator or provisioner may double click the left mouse button on a network device (e.g., 192.168.9.202, Figs. 11b or 11f) in the device list to cause a pop-up menu 1000 (Fig. 11h) to be displayed. The pop-up menu provides a list 1000a of available groups corresponding to the selected network device, and the administrator or provisioner may select one or more groups (e.g., Walmart-East, Walmart-West) from the list for which the user corresponding to profile will be authorized to access.

Each group may include one or more configured resources (e.g., SONET paths, VATM interfaces, ATM PVCs) within the network device, and the resources in each group may be related in some way. For instance, a group may include resources configured by a particular provisioner. As another example, a group may include configured resources purchased by a particular customer. For instance, Walmart Corporation may be a customer of a network service provider and each network device resource paid for / subscribed to by Walmart may be included in a Walmart group. In addition, if Walmart subscribes to a larger number of configured resources, the network service provider may create several groups within the same network device for Walmart, for example, Walmart-East may include network device resources associated with Walmart activities in the eastern half of the United States

and Walmart-West may include network device resources associated with Walmart activities in the western half of the United States. In addition, the network service provider may create a Walmart-Total group including all configured resources within the network device paid for by Walmart. Various users may be given access to one or more groups. For example, a Walmart employee responsible for network service in the eastern half of the United States may be given access to only the Walmart-East group while another higher level Walmart employee is given access to both the Walmart-East and Walmart-West groups. In addition, the same group name may be used in multiple network devices to simplify tracking. Through profiles multiple users may be given access to the same or different groups of configured resources within each network device, and users may be given access to multiple groups of configured resources in different network devices.

When an administrator or a provisioner configures a network device resource, they may assign that resource to a particular group. For example, when an administrator or provisioner configures one or more SONET paths, they may assign each SONET path to a particular group. Referring to Fig. 11i-11k, within a SONET Path configuration wizard 1002, an administrator or provisioner may select a SONET Path within the SONET path table 1002a and type in a group name in field 1002b or select a group name from a pop-up menu displayed when dropdown button 1002c is selected. When the administrator / provisioner selects OK button 1002d or Modify button 1002e, the NMS client sends the SONET path data to the NMS server. The NMS server uses this data to fill in a SONET path table (e.g., 600', Figs. 11w and 60g) in configuration database 42. A new row is added to the SONET path table for each newly configured SONET path, and data in existing rows are modified for modified SONET paths.

In addition, the NMS server searches a Managed Resource Group table 1008 (Figs. 11L and 11w) within the configuration database for a match with each assigned group name. If no match is found for a group name, indicating the group name represents a new group, then the NMS server adds a row to the Managed Resource Group table, and the NMS server assigns the group an LID (e.g., 1145) and inserts the LID into an LID column 1008a. The NMS server also inserts the Managed Device PID (e.g., 1) from column 983b in Managed Device table 983 (Figs. 11w and 60a) in the

configuration database into a column 1008b and inserts the group name in column 1008c.

The NMS server also uses the SONET path data from the NMS client to add a row in a Managed Resource Table 1007 (Figs. 11m and 11w) in configuration database 42 for each newly configured SONET path or to modify data in existing rows for modified SONET paths. The NMS server assigns an LID (e.g., 4443) to each row and inserts the assigned LID into a column 1007a. The NMS server then inserts the assigned SONET path LID (e.g., 901) from Path LID column 600a (Fig. 60g) in the SONET path table into a Resource LID column 1007b. The NMS server also inserts the assigned group LID (e.g., 1145) from column 1008a in Managed Resource Group table 1008 (Fig. 11L) into a managed resource group LID column 1007c.

Just as each SONET path may be assigned to a group, each other type of configured resource / manageable entity within the network device may be assigned to a group. For example, when an administrator or provisioner configures a virtual ATM (VATM) interface, they may also assign the VATM interface to a group. Referring to Fig. 11n, within an Add V-ATM Interface dialog box 1004, an administrator or provisioner may type in a group name in a field 1004a or select a group name from a pop-up menu displayed when expansion button 1004b is selected. As another example, when an administrator or provisioner configures an ATM PVC, they may assign the ATM PVC to a particular group. Referring to Fig. 11o, in a virtual connection wizard 1006, the administrator or provisioner may assign an ATM PVC to a group by typing in a group name in a field 1006a or by selecting a group name from a pop-up menu displayed when expansion button (e.g., Group List) 1006b is selected. Again, when the administrator or provisioner selects OK button 1004c (Fig. 11n) or Finish button 1006c (Fig. 11o), the NMS client sends the relevant data to the NMS server. The NMS server updates Virtual ATM Interface table 993 (Fig. 60j), a Virtual Connection table 994 (Fig. 60k), Virtual Link table 995 (Fig. 60L) and Cross-Connect table 996 (Fig. 60m), as described below, and similar to the actions taken for the configured SONET paths, the NMS server adds a row to Managed Resource Group table 1008 (Fig. 11L) for each new group and a row to Managed Resource table 1007 (Fig. 11m) for each new managed resource – that is, for each new VATM interface

and for each new ATM PVC. This same process may be used to add any manageable entity to a group.

Instead of using a Managed Resource Group table and a Managed Resource table, the configured network device resource tables (e.g., SONET path table, Virtual ATM IF table, etc.) could include a group name field. However, the Managed Resource Group adds a layer of abstraction, which may allow each configured resource to belong to multiple groups. Moreover, the Managed Resource table provides scalability and modularity by not being tied to a particular resource type. That is, the Managed Resource table will include a row for each different type of configured resource and if the network device is upgraded to include new types of configurable resources, they too may be added to the Managed Resource table without having to upgrade other processes. If each configurable resource is limited to belonging to only one group, then the Managed Resource Table 1007 (Fig. 11m) may include only Resource LID 1007b and not LID 1007a.

Referring again to Figs. 11b-11g, after adding or changing a user profile, the administrator or provisioner selects OK button 908t. Selection of the OK button causes the NMS client (e.g., NMS client 850a, Fig. 11w) to send the information provided in the dialog box (or boxes) to an NMS server (e.g., NMS server 851a), and the NMS server uses the received information to update various tables in NMS database 61. In one embodiment, for a newly added user, the NMS server assigns a unique logical identification number (LID) to the user and adds a new row in a User table 1010 (Figs. 11p and 11w) in the NMS database including the assigned LID 1010a and the username 1010b, password 1010c and group access level 1010d provided by the NMS client. For example, the NMS server may add a new row 1010e including an assigned user LID of 2012, a username of Dave, a password of Marble and a group access level of provisioner.

The NMS server also adds a row to a User Managed Device table 1012 (Figs. 11q and 11w) for each network device listed in the user profile. For each row, the NMS server assigns a user managed device LID (e.g., 7892) and inserts it in an LID column 1012a. The NMS server also inserts a user LID 1012b, a host LID 1012c, a retry value 1012d and a timeout value 1012e. The inserted retry and timeout values are

from the user profile information sent from the NMS client. The user LID 1012b includes the previously assigned user LID (e.g., 2012) from column 1010a of User Table 1010. The host LID is retrieved from an Administration Managed Device table 1014 (Figs. 11r and 11w).

The Administration Managed Device table includes a row for each network device (i.e., managed device) in the telecommunications network. To add a network device to the network, an administrator selects an Add Device option in a pop-up menu 898c (Fig. 6a) in GUI 895 to cause dialog box 1013 (Fig. 11s) to be displayed. The administrator enters the intended IP address or DNS name (e.g., 192.168.9.202) of the new network device into a device host field 1013a and may also enter a device port (e.g., 1521) into a device port field 1013b. The administrator also adds SNMP retry 1013c and timeout 1013d values, which may be overridden later by values supplied within each user profile. In addition, the administrator adds a password for each user access level. In one embodiment, the administrator adds an administrator password 1013e, a provisioner password 1013f and a viewer password 1013g for the managed device.

The Administration Managed Device table, therefore, provides a centralized set of device records shared by all NMS servers, and since the records are centralized, the Administration Managed Device table facilitates centralized changes to the devices in the network. For example, a network device may be added to the network by adding a record and removed from the network by deleting a record. As another example, a network device's parameters (e.g., IP address) may be changed by modifying data in a record. Because the changes are made to centralized records accessed by all NMS servers, no change notifications need to be sent and the NMS servers may automatically receive the changed data during the next access of the table. Alternatively, the NMS server that makes a change to the central database may send notices out to each connected NMS client and other NMS servers in the network.

For newly added devices, after the information is input in the dialog box, the administrator selects an Add button 1013h causing the NMS client to send the data from the dialog box to the NMS server. Similarly, for changes to device data, after the information is changed in the dialog box, the administrator selects an OK button

1013i to cause the NMS client to send the data from the dialog box to the NMS server. For new devices, the NMS server uses the received information to add a row to Administration Managed Device table 1014 in NMS database 61, and for existing devices, the NMS server uses the received information to update a previously entered row in the Administration Managed Device table. For each managed device / row, the NMS server assigns a host LID (e.g., 9046) and inserts it in LID column 1014a.

When the NMS server adds a new row to the User Managed Device table 1012 (Fig. 11q), corresponding to a managed device in a user profile, the NMS server searches column 1014b in the Administration Managed Device table 1014 for a host address matching the IP address (e.g., 192.168.9.202) provided in the user profile information sent from the NMS client. When a match is found, the NMS server retrieves the host LID (e.g., 9046) from column 1014a and inserts it in host LID column 1012c in the User Managed Device table.

After receiving user profile information from an NMS client, the NMS server also updates a User Resource Group Map table 1016 (Figs. 11t and 11w) in NMS database 61. For each group identified in the user profile information – one or more groups may be selected in each Group List dialog box 1000 associated with each network device in the user profile – the NMS server adds a row to the User Resource Group Map table. The NMS server assigns an LID (e.g., 8086) for each row and inserts the LID in a column 1016a. The NMS server then inserts the User LID (e.g., 2012) into User LID column 1016b from User table 1010 column 1010a corresponding to the user profile. In addition, the NMS server inserts a User Resource Group LID into column 1016c.

For each group name received by the NMS server, the NMS server searches a User Resource Group table 1018 (Figs. 11u and 11w), group name column 1018c, for a match. If a match is not found, then the group is a new group, and the NMS server adds a row to the User Resource Group table. The NMS server assigns an LID (e.g., 1024) to each row and inserts the assigned LID into an LID column 1018a. This User Resource Group LID is also added to column 1016c in the User Resource Group Map table 1016 (Fig. 11t). Within the User Resource Group table 1018 (Fig. 11u), the NMS server also inserts the network device's host LID in a column 1018b from

Administration Managed Device table 1014 (Fig. 11r), column 1014a, and the NMS server inserts the group name (e.g., Walmart-East) in column 1018c. Through the group name, the User Resource Group table in the NMS database provides for dynamic binding with the Managed Resource Group table 1008 (Fig. 11L) in the configuration database, as described below.

After a user's profile is created, the user may log in through an NMS client (e.g., 850a, Fig. 11w) by typing in their username and password. The NMS client then sends the username and password to an NMS server (e.g., 851a), and in response, the NMS server sends a query to NMS database 61 to search User table 1010 (Fig. 11p) column 1010b for a username matching the username provided by the NMS client. If the username is not found, then the user is denied access. If the username is found, then, for additional security, the NMS server may compare the password provided by the NMS client to the password stored in column 1010c of the User table. If the passwords do not match, then the user is denied access. If the passwords match, then the NMS server creates a user profile logical managed object (LMO).

In one embodiment, the user profile LMO is a JAVA object and a JAVA persistence layer within the NMS server creates the user profile LMO. For each persistent JAVA class / object, metadata is stored in a class table 1020 (Fig. 11w) within the NMS database. Thus, the JAVA persistence layer within the NMS server begins by retrieving metadata from the class table in the NMS database corresponding to the user profile LMO. The metadata may include simple attributes and association attributes.

Referring to Fig. 11v, the metadata for a user profile LMO 1022 includes three simple attributes – username 1022a, password 1022b and group access level 1022c – and two association attributes – resource group maps 1022d and managed devices 1022e. The NMS server inserts the username (e.g., Dave), password (e.g., Marble) and group access level (e.g., provisioner) retrieved from the User table 1010 into the user profile LMO 1024 (Fig. 11w) being created. The managed devices association attribute 1022e causes the NMS server to create a user managed device properties LMO 1026 for each network device in the user's profile.

The NMS server first retrieves metadata from class table 1020 associated with the user managed device properties LMO 1026. The metadata includes two simple attributes (retry 1026b and timeout 1026c) and one association attribute (managed device 1026a). The metadata causes the NMS server to search User Managed Device table 1012 (Fig. 11q) column 1012b for a user LID (e.g., 2012) corresponding to the user LID in column 1010a (Fig. 11p) of User table 1010 in a row 1010e associated with the username and password received from the NMS client. For each row in the User Managed Device table having the matching user LID (e.g., 2012), the NMS server creates a user managed device properties LMO 1026 and inserts the retry value from column 1012d as the retry simple attribute 1026b and the timeout value from column 1012e as the timeout simple attribute 1026c.

In response to the managed device associated attribute, the NMS server retrieves metadata from class table 1020 associated with administration managed device properties LMO 1028. The metadata includes a list of simple attributes including host address 1028a, port address 1028b, SNMP retry value 1028c, SNMP timeout value 1028d and a database port address 1028e for connecting to the configuration database within the network device. The metadata also includes simple attributes corresponding to passwords for each of the possible group access levels, for example, an administrator password 1028f, a provisioner password 1028g and a viewer password 1028h.

The NMS server uses the host LID (e.g., 9046) from column 1012c in the User Managed Device table (Fig. 11q) as a primary key to locate the row (e.g., 1014c, Fig. 11r) in the Administration Managed Device table 1014 corresponding to the network device. The NMS server uses the data in this table row to insert values for the simple attributes in the Administration Managed Device LMO 1028. For example, a host address of 192.168.9.202 and a port address of 1521 may be inserted. The NMS server also selects a password corresponding to the user's group access level. For instance, if the user's group access level is provisioner, then the NMS server inserts the provisioner password of, for example, team2, from column 1014d into the Administration Managed Device LMO.

The NMS server then inserts the newly created Administration Managed Device LMO 1028 into the corresponding User Managed Device Properties LMO 1026, and the NMS server also inserts each newly created User Managed Devices Properties LMO 1026 into User Profile LMO 1022. Thus, the information necessary for connecting to each network device listed in the user profile is made available within user LMO 1022.

The resource group maps association attribute 1022d (Fig. 11v) within user LMO 1022 causes the NMS server to create a user resource group map LMO 1030 for each group in the user's profile. The user resource group map LMO 1030 includes one simple attribute -- user profile 1030a -- and one association attribute – user resource group 1030b. The NMS server inserts the user LID (e.g., 2012) corresponding to the user LID in column 1010a (Fig. 11p) in User table 1010 associated with the username, password and group access level received from the NMS client.

In response to user resource group associated attribute 1030b, the NMS server creates a User Resource Group LMO 1032. The NMS server begins by retrieving metadata from class table 1020 corresponding to the User Resource Group LMO. The metadata includes three simple attributes: host address 1032a, port address 1032b and group name 1032c. The NMS server searches User Resource Group Map table 1016 (Fig. 11t) for the user LID (e.g., 2012) corresponding to the username and password received from the NMS client. The NMS server then uses the corresponding user resource group LID (e.g., 1024) from column 1016c as a primary key to locate a row (e.g., 1018d, Fig. 11u) in User Resource Group table 1018. The NMS server inserts the group name (e.g., Walmart-East) from the located row in User Resource Group table 1018 as simple attribute 1032c in user resource group LMO 1032. The NMS server then uses the host LID (e.g., 9046) from the located row to search column 1014a in the Administration Managed Device table 1014 (Fig. 11r) for a match. Once a match is found, the NMS server uses data in the located row (e.g., 1014c) to insert the host address (e.g., 192.168.9.202) from column 1014b as simple attribute 1032a and the port address (e.g., 1521) from column 1014e as simple attribute 1032b in user resource group LMO 1032. The NMS server then inserts the user resource group LMO 1032 into the user resource group map LMO 1030, and the NMS server inserts each of the user resource group map LMOs 1030 into the user profile LMO 1022.

Thus, the data (e.g., host and port address and group name) required to locate each group included in the user profile is inserted within user profile LMO 1022.

The NMS server sends data from the user profile LMO to the NMS client to allow the NMS client to present the user with a graphical user interface such as GUI 895 shown in Fig. 4a. If the user selects one of the network devices listed in navigation tree 898, the NMS server retrieves the group level access (e.g., provisioner) and the password (e.g., team2) corresponding to that group level access from the user profile LMO and then connects to the selected network device. The NMS server then retrieves the network device's physical data as described below under the heading "NMS Server Scalability."

Alternatively, a more robust set of data may be sent from the NMS server to the NMS client such that for each transaction issued by the NMS client, the data provided with the transaction eliminates the need for the NMS server to access the user profile LMO in its local memory. This reduces the workload of the NMS server, which will likely be sent transactions from many NMS clients. In one embodiment, the NMS server may send the NMS client the entire user profile LMO. Instead, the server may create a separate client user profile LMO that may present the data in a format expected by the NMS client and perhaps include only some of the data from the user profile LMO stored locally to the NMS server. In the preferred embodiment, the client user profile LMO includes at least data corresponding to each device in the user profile and each group selected within the user profile for each device. If the user selects one of the network devices listed in navigation tree 898, the NMS client includes the selected network device's IP address, the password corresponding to the user's group access level and the database port number in the "Get Network Device" transaction sent to the NMS server. The NMS server uses this information to connect to the network device and return the network device's physical data to the NMS client.

If the user selects a tab in configuration status window 897 that includes logical data corresponding to configured network device resources (e.g., SONET Paths tab 942 (Fig. 5q), ATM Interfaces tab 946 (Fig. 5r), Virtual ATM Interfaces tab 947 (Fig. 5s), Virtual Connections tab 948 (Fig. 5z)), then the NMS server searches the user profile LMO for group names corresponding to the selected network device or the NMS

client provides the group names in the transaction. The NMS server then retrieves data from the selected network device for configured resources corresponding to each group name and the selected tab. If no group names are listed, the NMS server may retrieve data for all configured resources corresponding to the selected tab.

For example, if a user selects SONET Paths tab 942 (Fig. 5q), then the NMS server searches the user profile LMO for all group names corresponding to the selected network device (e.g., Walmart-East) or the NMS client provides all group names (e.g., Walmart-East) corresponding to the selected network device to the NMS server as part of the "Get SONET paths" transaction. The NMS server then dynamically issues a where clause such as "where SONET path is in group Walmart-East". This causes group name column 1008c in the Managed Resource Group table 1008 (Fig. 11L) in the network device's configuration database 42 to be searched for a match with the group name of Walmart-East. Additional where clauses may be dynamically issued corresponding to other group names found in the user profile LMO. If no match is found for a group name in column 1008c, then the NMS server simply returns an empty set to the NMS client. If a match is found for a group name (e.g., Walmart-East), then the NMS server retrieves the managed resource group LID (e.g., 1145) from column 1008a in the same row (e.g., row 1008d) as the matching group name.

The NMS server then searches column 1007c in the Managed Resource table 1007 (Fig. 11m) for one or more matches with the retrieved managed resource group LID (e.g., 1145). As described above, the Managed Resource Table includes one row for each configured network device resource in a particular group. For each match found for the retrieved managed resource group LID (e.g., 1145), the NMS server uses the resource LID (e.g., 901) from column 1007b as a primary key to a row in a table including the data corresponding to the configured resource. In this example, a resource LID of 901 corresponds to a row in SONET Path Table 600' (Fig. 60g). Since the user selected the SONET Paths tab, the NMS server retrieves the data in the corresponding row and sends it to the NMS client. The NMS client uses the data to update graphical user interface (GUI) tables 985 in local memory 986, which causes GUI 895 to display the SONET path to the user. Other SONET paths may also be included in the group Walmart-East, and those would be similarly located and retrieved by the NMS server and sent to the NMS client for display to the user.

Since each group may include different types of configured resources, the NMS server may locate configured resources other than SONET paths, for example, VATMs or ATM PVCs, in Managed Resource table 1007. If configured resources are found that do not correspond to the tab selected by the user, the NMS server does not retrieve the associated data or send it to the NMS client. The NMS server follows a similar process if the user selects another tab including logical data, for example, ATM Interfaces tab 946 (Fig. 5r), Virtual ATM Interfaces tab 947 (Fig. 5s) or Virtual Connections tab 948 (Fig. 5z). Although the above discussion has used SONET paths, VATM interfaces and ATM PVCs as examples of configurable resources that may be included in a group, other configurable resources may also be included, for example, configurable resources corresponding to different layer one or upper layer network protocols (e.g., Ethernet, MPLS, Frame Relay, IP).

When data is stored in tables within the same database, references from one table to another may provide a direct binding and referential integrity may be maintained by only deleting the upper most record – that is, not leaving any dangling records. Referential integrity prevents references from being orphaned, which may lead to data loss or other more severe problems, such as a system crash. In the current embodiment, tables are stored across multiple databases. Certain tables are stored in NMS database 61 and certain other tables are stored in the configuration database within each network device in the network. Direct binding between tables cannot be maintained since a database may be removed or a record deleted without maintaining referential integrity. To address this issue, group names are used to provide a "dynamic binding" between the User Resource Group table 1018 (Fig. 11u) in the NMS database and the Managed Resource Group table 1008 (Fig. 11L) in each configuration database. Since there is no direct binding, if a group name is not found in the Managed Resource Group table, the NMS server simply returns an empty set and no data is lost or other more serious problems caused. If the group name is later added to the Managed Resource Group table, then through dynamic binding, it will be found.

Through a user profile, a user may log-on to the network with a single, secure username and password through any NMS client, access any network device in their

user profile and access configured resources corresponding to groups in their user profile. Since the tables including the data necessary for the creation of user profile LMOs are stored in the NMS database, any NMS server capable of connecting to the NMS database – that is, any NMS server in the network -- may access the tables and generate a user LMO. As a result, users may log-on with a single, secure username and password through any NMS client that may be connected to an NMS server capable of connecting to the NMS database. Essentially, users may log on through any computer system / workstation (e.g., 984, Fig. 11w) on which an NMS client is loaded or remotely through internet web access to an NMS client within the network and gain access to the network devices listed in their user profile. Thus, each user need only remember a single username and password to configure / manage any of the network devices listed in their user profile or any of the resources included within groups listed in their user profile through any NMS client in the network.

In addition, user profiles provide a level of indirection to better protect the passwords used to access each network device. For example, access to the passwords may be limited to only those users capable of adding network devices to the network, for example, users with the administrator group access level. Other users would not see the passwords since they are automatically added to their user profile LMO, which is not accessible by users. The level of indirection provided by user profiles also allows network device passwords to be easily changed across the entire network. Periodically the passwords for access to the network devices in a network may be changed for security. The network device passwords may be quickly changed in the Administration Managed Device table 1014 (Fig. 11r), and due to the use of profiles, each user does not need to be notified of the password changes. The new passwords will be utilized automatically each time users log in. This provides for increased scalability since thousands of users will not need to be notified of the new passwords. Moreover, if a rogue user is identified, they can be quickly prevented from further access to the network through any NMS client by simply changing the user's username and/or password in the user's profile or by deleting the user's profile. Changing the username and/or password in the user profile would cause the NMS server to change the data in user table 1010 (Fig. 11p), and deleting a user profile would cause the NMS server to remove the corresponding row in the User table. In either case, the user would no longer be able to log in.

User profiles and group names also simplify network management tasks. For example, if an administrator adds a newly configured resource to a group, all users having access to that group will automatically be able to access the newly configured resource. The administrator need not send out a notice or take other steps to update each user.

Group names in a user profile define what the user can view. For instance, one customer may not view the configured resources subscribed for by another customer if their resources are assigned to different groups. Thus, groups allow for a granular way to "slice" up each network device according to its resources.

The user access level in a user profile determines how the NMS server behaves and affects what the user can do. For example, the viewer user access level provides the user with read-only capability and, thus, prevents the NMS server from modifying data in tables. In addition, the user access level may be used to restrict access – even read access -- to certain tables or columns in certain tables.

Network Device Power-Up:

Referring again to Fig. 1, on power-up, reset or reboot, the processor on each board (central processor and each line card) downloads and executes boot-strap code (i.e., minimal instances of the kernel software) and power-up diagnostic test code from its local memory subsystem. After passing the power-up tests, processor 24 on central processor 12 then downloads kernel software 20 from persistent storage 21 into non-persistent memory in memory subsystem 28. Kernel software 20 includes operating system (OS), system services (SS) and modular system services (MSS).

In one embodiment, the operating system software and system services software are the OSE operating system and system services from Enea OSE Systems, Inc. in Dallas, Texas. The OSE operating system is a pre-emptive multi-tasking operating system that provides a set of services that together support the development of distributed applications (i.e., dynamic loading). The OSE approach uses a layered architecture that builds a high level set of services around kernel primitives. The operating system, system services, and modular system services provide support for

the creation and management of processes; inter-process communication (IPC) through a process-to-process messaging model; standard semaphore creation and manipulation services; the ability to locate and communicate with a process regardless of its location in the system; the ability to determine when another process has terminated; and the ability to locate the provider of a service by name.

These services support the construction of a distributed system wherein applications can be located by name and processes can use a single form of communication regardless of their location. By using these services, distributed applications may be designed to allow services to transparently move from one location to another such as during a fail over.

The OSE operating system and system services provide a single inter-process communications mechanism that allows processes to communicate regardless of their location in the system. OSE IPC differs from the traditional IPC model in that there are no explicit IPC queues to be managed by the application. Instead each process is assigned a unique process identification that all IPC messages use. Because OSE IPC supports inter-board communication the process identification includes a path component. Processes locate each other by performing an OSE Hunt call on the process identification. The Hunt call will return the Process ID of the process that maps to the specified path/name. Inter-board communication is carried over some number of communication links. Each link interface is assigned to an OSE Link Handler. The path component of a process path/name is the concatenation of the Link Handler names that one must transverse in order to reach the process.

In addition, the OSE operating system includes memory management that supports a "protected memory model". The protected memory model dedicates a memory block (i.e., defined memory space) to each process and erects "walls" around each memory block to prevent access by processes outside the "wall". This prevents one process from corrupting the memory space used by another process. For example, a corrupt software memory pointer in a first process may incorrectly point to the memory space of a second processor and cause the first process to corrupt the second processor's memory space. The protected memory model prevents the first process with the corrupted memory pointer from corrupting the memory space or block assigned to the

second process. As a result, if a process fails, only the memory block assigned to that process is assumed corrupted while the remaining memory space is considered uncorrupted.

The modular software architecture takes advantage of the isolation provided to each process (e.g., device driver or application) by the protected memory model. Because each process is assigned a unique or separate protected memory block, processes may be started, upgraded or restarted independently of other processes.

Referring to Fig. 12a, the main modular system service that controls the operation of computer system 10 is a System Resiliency Manager (SRM). Also within modular system services is a Master Control Driver (MCD) that learns the physical characteristics of the particular computer system on which it is running, in this instance, computer system 10. The MCD and the SRM are distributed applications. A master SRM 36 and a master MCD 38 are executed by central processor 12 while slave SRMs 37a-37n and slave MCDs 39a-39n are executed on each board (central processor 12 and each line card 16a-16n). The SRM and MCD work together and use their assigned view ids and APIs to load the appropriate software drivers on each board and to configure computer system 10.

Also within the modular system services is a configuration service program 35 that downloads a configuration database program 42 and its corresponding DDL file from persistent storage into non-persistent memory 40 on central processor 12. In one embodiment, configuration database 42 is a Polyhedra database from Polyhedra, Inc. in the United Kingdom.

Hardware Inventory and Set-Up:

Master MCD 38 begins by taking a physical inventory of computer system 10 (over the I^2C bus) and assigning a unique physical identification number (PID) to each item. Despite the name, the PID is a logical number unrelated to any physical aspect of the component being numbered. In one embodiment, pull-down/pull-up resistors on the chassis mid-plane provide the number space of Slot Identifiers. The master MCD may read a register for each slot that allows it to get the bit pattern produced by these resistors. MCD 38 assigns a unique PID to the chassis, each shelf in the chassis, each

slot in each shelf, each line card 16a-16n inserted in each slot, and each port on each line card. (Other items or components may also be inventoried.)

Typically, the number of line cards and ports on each line card in a computer system is variable but the number of chassis, shelves and slots is fixed. Consequently, a PID could be permanently assigned to the chassis, shelves and slots and stored in a file. To add flexibility, however, MCD 38 assigns a PID even to the chassis, shelves and slots to allow the modular software architecture to be ported to another computer system with a different physical construction (i.e., multiple chassis and / or a different number of shelves and slots) without having to change the PID numbering scheme.

Referring to Figs. 12a-12c, for each line card 16a-16n in computer system 10, MCD 38 communicates with a diagnostic program (DP) 40a-40n being executed by the line card's processor to learn each card's type and version. The diagnostic program reads a line card type and version number out of persistent storage, for example, EPROM 42a-42n, and passes this information to the MCD. For example, line cards 16a and 16b could be cards that implement Asynchronous Transfer Mode (ATM) protocol over Synchronous Optical Network (SONET) protocol as indicated by a particular card type, e.g., 0XF002, and line card 16e could be a card that implements Internet Protocol (IP) over SONET as indicated by a different card type, e.g., 0XE002. In addition, line card 16a could be a version three ATM over SONET card meaning that it includes four SONET ports 44a-44d each of which may be connected to an external SONET optical fiber that carries an OC-48 stream, as indicated by a particular port type 00620, while line card 16b may be a version four ATM over SONET card meaning that it includes sixteen SONET ports 46a-46f each of which carries an OC-3 stream as indicated by a particular port type, e.g., 00820. Other information is also passed to the MCD by the DP, for example, diagnostic test pass / fail status. With this information, MCD 38 creates card table (CT) 47 and port table (PT) 49 in configuration database 42. As described below, the configuration database copies all changes to an NMS database. If the MCD cannot communicate with the diagnostic program to learn the card type and version number, then the MCD assumes the slot is empty.

Even after initial power-up, master MCD 38 will continue to take physical inventories to determine if hardware has been added or removed from computer system 10. For example, line cards may be added to empty slots or removed from slots. When changes are detected, master MCD 38 will update CT 47 and PT 49 accordingly.

For each line card 16a-16n, master MCD 38 searches a physical module description (PMD) file 48 in memory 40 for a record that matches the card type and version number retrieved from that line card. The PMD file may include multiple files. The PMD file includes a table that corresponds card type and version number with name of the mission kernel image executable file (MKI.exe) that needs to be loaded on that line card. Once determined, master MCD 38 passes the name of each MKI executable file to master SRM 36. Master SRM 36 requests a bootserver (not shown) to download the MKI executable files 50a-50n from persistent storage 21 into memory 40 (i.e., dynamic loading) and passes each MKI executable file 50a-50n to a bootloader (not shown) running on each board (central processor and each line card). The bootloaders execute the received MKI executable file.

Once all the line cards are executing the appropriate MKI, slave MCDs 39a-39n and slave SRMs 37a-37n on each line card need to download device driver software corresponding to the particular devices on each card. Referring to Fig. 13a, slave MCDs 39a-39n search PMD file 48 in memory 40 on central processor 12 for a match with their line card type and version number. Just as the master MCD 36 found the name of the MKI executable file for each line card in the PMD file, each slave MCD 39a-39n reads the PMD file to learn the names of all the device driver executable files associated with each line card type and version. The slave MCDs provide these names to the slave SRMs on their boards. Slave SRMs 37a-37n then download and execute the device driver executable files (DD.exe) 56a-56n from memory 40. As one example, one port device driver 43a-43d may be started for each port 44a-44d on line card 16a. The port driver and port are linked together through the assigned port PID number.

In order to understand the significance of the PMD file (i.e., metadata), note that the MCD software does not have knowledge of board types built into it. Instead, the MCD parameterizes its operations on a particular board by looking up the card type

and version number in the PMD file and acting accordingly. Consequently, the MCD software does not need to be modified, rebuilt, tested and distributed with new hardware. The changes required in the software system infrastructure to support new hardware are simpler modify logical model 280 (Fig. 3a) to include: a new entry in the PMD file (or a new PMD file) and, where necessary, new device drivers and applications. Because the MCD software, which resides in the kernel, will not need to be modified, the new applications and device drivers and the new DDL files (reflecting the new PMD file) for the configuration database and NMS database are downloaded and upgraded (as described below) without re-booting the computer system.

Network Management System (NMS):

Referring to Fig. 13b, as described above, a user / network administrator of computer system 10 works with network management system (NMS) software 60 to configure computer system 10. In the embodiment described below, NMS 60 runs on a personal computer or workstation 62 and communicates with central processor 12 over Ethernet network 41 (out-of-band). Instead, the NMS may communicate with central processor 12 over data path 34 (Fig. 1, in-band). Alternatively (or in addition as a back-up communication port), a user may communicate with computer system 10 through a console interface / terminal (840, Fig. 2a) connected to a serial line 66 connecting to the data or control path using a command line interface (CLI) protocol. Instead, NMS 60 could run directly on computer system 10 provided computer system 10 has an input mechanism for the user.

During installation, an NMS database 61 is established on, for example, work-station 62 using a DDL executable file corresponding to the NMS database. The DDL file may be downloaded from persistent storage 21 in computer system 10 or supplied separately with other NMS programs as part of an NMS installation kit. The NMS database mirrors the configuration database through an active query feature (described below). In one embodiment, the NMS database is an Oracle database from Oracle Corporation in Boston, Massachusetts.

The NMS and central processor 12 pass control and data over Ethernet 41 using, for example, the Java Database Connectivity (JDBC) protocol. Use of the JDBC protocol

allows the NMS to communicate with the configuration database in the same manner that it communicates with its own internal storage mechanisms, including the NMS database. Changes made to the configuration database are passed to the NMS database to ensure that both databases store the same data. This synchronization process is much more efficient, less error-prone and timely than older methods that require the NMS to periodically poll the network device to determine whether configuration changes have been made. In these systems, NMS polling is unnecessary and wasteful if the configuration has not been changed. Additionally, if a configuration change is made through some other means, for example, a command line interface, and not through the NMS, the NMS will not be updated until the next poll, and if the network device crashes prior to the NMS poll, then the configuration change will be lost. In computer system 10, however, command line interface changes made to configuration database 42 are passed immediately to the NMS database through the active query feature ensuring that the NMS, through both the configuration database and NMS database, is immediately aware of any configuration changes.

Asynchronously Providing Network Device Management Data:

Typically, work-station 62 (Fig. 13b) is coupled to many network computer systems, and NMS 60 is used to configure and manage each of these systems. In addition to configuring each system, the NMS also interprets management data gathered by each system relevant to each system's network accounting data, statistics, security and fault logging (or some portion thereof) and presents this to the user. In current systems, two distributed carefully synchronized processes are used to move data from a network system/device to the NMS. The processes are synchronized with each other by having one or both processes maintain the state of the other process. To avoid the problems associated with using two synchronized processes, in the present invention, internal network device management subsystem processes are made asynchronous with external management processes. That is, neither the internal nor external processes maintain each other's state and all processes operate independently of the other processes. This also minimizes or prevents data loss (i.e., lossless system), which is especially important for revenue generating accounting systems.

In addition, instead of having the NMS interpret each network device's management data in the same fashion, flexibility is added by having each system send the NMS (e.g., data collector server 857, Fig. 2a) class files 410 including compiled source code indicating how its management data should be interpreted. Thus, the NMS effectively "learns" how to process (and perhaps display) management data from the network device via the class file. Through the reliable File Transfer Protocol (FTP), management subsystem processes 412 (Fig. 13b) running on central processor 12 push data summary files 414 and binary data files 416 to the NMS. Each data summary file indicates the name of the class file the NMS should use to interpret a corresponding binary data file. If the computer system has not already done so, it pushes the class file to the NMS. In one embodiment, the management subsystem processes, class files and NMS processes are JAVA programs, and JAVA Reflection is used to dynamically load the data-specific application class file and process the data in the binary data file. As a result, a new class file can be added or updated on a network device without having to reboot or upgrade the network device or the NMS. The computer system simply pushes the new class file to the NMS. In addition, the NMS can use different class files for each network device such that the data gathered on each device can be particularized to each device.

Referring to Fig. 13c, in one embodiment, the management subsystem 412 (Fig. 13b) is broken into two pieces: a usage data server (UDS) 412a and a file transfer protocol (FTP) client 412b. The UDS is executed on internal processor control card 542a (see also Figs. 41b and 42) while the FTP client is executed on external processor control card 542b (see also Figs. 41a and 42). Alternatively, in a network device with one processor control card or a central processor control card, both the UDS and FTP client may be executed on that one card. When each device driver, for example, SONET driver 415a-415n and ATM driver 417a-417n (only SONET driver 415a and ATM driver 417a are shown for convenience and it is to be understood that multiple drivers may be present on each card), within network device 540 is built, it links in a usage data monitoring library (UDML).

When device drivers are first started, upgraded or re-booted, the device driver makes a call into the UDML to notify the UDML as to which statistical data the device driver is able to gather. For example, an ATM device driver may be able to gather

virtual circuit (VC) accounting statistics and Virtual ATM (VATM) interface statistics while a SONET device driver may be able to gather SONET statistics. The device driver then makes a call into the UDML to notify the UDML as to each interface (including virtual circuits) for which the device driver will be gathering data and the types of data the device driver will provide for each interface.

The UDML sends a registration packet to the UDS providing one or more string names corresponding to the types of data that the UDML will send to the UDS. For example, for ATM drivers the UDML may register "Acct_PVC" to track permanent virtual circuit statistics, "Acct_SVC" to track soft permanent virtual circuit statistics, "Vir_Intf" to track quality of service (QoS) statistics corresponding to virtual interfaces, and "Bw_Util" to track bandwidth utilization. As another example, for SONET drivers the UDML may register "Section" to track section statistics, "Line" to track line statistics and "Path" to track path statistics. The UDML need only register each string name with the UDS once, for example, for the first interface registered, and not for each interface since the UDML will package up the data from multiple interfaces corresponding to the same string name before sending the data with the appropriate string name to the UDS.

The UDML includes a polling timer to cause each driver to periodically poll its hardware for "current" statistical/accounting data samples 411a. The current data samples are typically gathered on a frequent interval of, for example, 15 minutes, as specified by the polling timer. The UDML also causes each driver to put the binary data in a particular format, time stamp the data and store the current data sample locally. When a current data sample for each interface managed by the device driver and corresponding to a particular string name is stored locally, the UDML packages all of the current data samples corresponding to the same string name into one or more packets containing binary data and sends the packets to the UDS with the registered string name.

In addition, the UDML adds each gathered current data sample 411a to a local data summary 411b. The UDML clears the data summary periodically, for example, every twenty-four hours, and then adds newly gathered current data samples to the cleared

data summary. Thus, the data summary represents an accumulation of current data samples gathered over the period (e.g., 24 hours).

The UDS maintains a list of UDMLs expected to send current data samples and data summaries corresponding to each string name. For each poll, the UDS combines the data sent from each UDML with the same string name into a common binary data file (e.g., binary data files 416a-416n) associated with that string name in non-volatile memory, for example, a hard drive 421 located on internal control processor 542a. When all UDMLs in the list corresponding to a particular string name have reported their current data samples or data summaries, the UDS closes the common data file, thus ending the data collecting period. Preferably, the data is maintained in binary form to keep the data files smaller than translating it into other forms such as ASCII. Smaller binary files require less space to store and less bandwidth to transfer.

If after a predetermined period of time has passed, for example, 5 minutes, one or more of the UDMLs in a list has not sent binary data with the corresponding string name, the UDS closes the common data file, ending the data collecting period. The UDS then sends a notice to the non-responsive UDML(s). The UDS will repeat this sequence a predetermined number of times, for example, three, and if no binary data with the corresponding string name is received, the UDS will delete the UDML(s) from the list and send a trap to the NMS indicating which specific UDML is not responsive. As a result, maintaining the list of UDMLs that will be sending data corresponding to each string name allows the UDS to know when to close each common data file and also allows the UDS to notify the NMS when a UDML becomes non-responsive. This provides for increased availability including fault tolerance – that is, a fault on one card or in one application cannot interrupt the statistics gathering from each of the other cards or other applications on one card -- and also including hot swapping where a card and its local UDMLs may no longer be inserted within the network device.

Since a large number of UDMLs may be sending data to the UDS, the potential exists for the data transfer rate to the UDS to be larger than the amount of data that the UDS can process and larger than local buffering can support. Such a situation may result in lost data or worse, for example, a network device crash. A need exists, therefore, to

be able to "throttle" the amount of data being sent from the UDMLs to the UDS depending upon the current backlog of data at the UDS.

In one embodiment, the UDML is allowed to send up to a maximum number of packets to the UDS before the UDML must wait for an acknowledge (ACK) packet from the UDS. For example, the UDML may be allowed to send three packets of data to the UDS and in the third packet the UDML must include an acknowledge request. Alternatively, the UDML may follow the third packet with a separate packet including an acknowledge request. Once the third packet is sent, the UDML must delay sending any additional packets to the UDS until an acknowledge packet is received from the UDS. The UDML may negotiate the maximum number of packets that can be sent in its initial registration with the UDS. Otherwise, a default value may be used.

Many packets may be required to completely transfer a binary current data sample or data summary to the UDS. Once the acknowledge packet is received, the UDML may again send up to the maximum number (e.g., 3) of packets to the UDS again including an acknowledge request in the last packet. Requiring the UDML to wait for an acknowledge packet from the UDS, allows the UDS to throttle back the data received from UDMLs when the UDS has a large backlog of data to process.

A simple mechanism to accomplish this throttling is to have the UDS send an acknowledge packet each time it processes a packet containing an acknowledge request. Since the UDS is processing the packet that is a good indication that it is steadily processing packets. If the number of packets received by the UDS is large, it will take longer to process the packets and, thus, longer to process packets containing acknowledge requests. Thus, the UDMLs must wait longer to send more packets. On the other hand, if the number of packets is small, the UDS will quickly process each packet received and more quickly send back the acknowledge request and the UDMLs will not have to wait as long to send more packets.

Instead of immediately returning an acknowledge packet when the UDS processes a packet containing an acknowledge request, the UDS may first compare the number of packets waiting to be processed against a predetermined threshold. If the number of

packets waiting to be processed is less than the predetermined threshold, then the UDS immediately sends the acknowledge packet to the UDML. If the number of packets waiting to be processed is more than the predetermined threshold, then the UDS may delay sending the acknowledge packet until enough packets have been processed that the number of packets waiting to be processed is reduced to less than the predetermined threshold. Instead, the UDS may estimate the amount of time that it will need to process enough packets to reduce the number of packets waiting to be processed to less than the threshold and send an acknowledge packet to the UDML including a future time at which the UDML may again send packets. In other words, the UDS does not wait until the backlog is diminished to notify the UDMLs but instead notifies the UDMLs prior to reducing the backlog and based on an estimate of when the backlog will be diminished.

Another embodiment for a throttling mechanism requires polls for different statistical data to be scheduled at different times to load balance the amount of statistical traffic across the control plane. For example, the UDML for each ATM driver polls and sends data to the UDS corresponding to PVC accounting statistics (i.e., Acct_PVC) at a first time, the UDML for each ATM driver polls and sends data to the UDS corresponding to SPVC accounting statistics (i.e., Acct_SPVC) at a second time, and the UDML for each ATM driver and each SONET driver polls and sends data to the UDS corresponding to other statistics at other times. This may be accomplished by having multiple polling timers within the UDML corresponding to the type of data being gathered. Load balancing and staggered reporting provides distributed data throttling which may smooth out control plane bandwidth utilization (i.e., prevent large data bursts) and reduce data buffering and data loss.

Referring to Fig. 13d, instead of having each device driver on a card package the binary data and send it to the UDS, a separate, low priority packaging program (PP) 413a-413n may be resident on each card and responsible for packaging the binary statistical management data from each device driver and sending it to the UDS. Running the PP as a lower priority program ensures that processor cycles are not taken away from time-critical processes. Load balancing and staggered reporting may still be accomplished by having each PP send acknowledge requests in the last of a

predetermined number of packets and wait for the UDS to send an acknowledge packet as described above.

As mentioned, the UDML causes the device driver to periodically gather the current statistical management data samples for each interface and corresponding to each string name. The period may be relatively frequent, for example, every 15 minutes. In addition, the UDML causes the device driver or separate packaging program to add the current data sample to a data summary corresponding to the same string name each time a current data sample is gathered. The UDML clears the data summary periodically, for example, every twenty-four hours. To reduce bandwidth utilization, the data summary and corresponding string name is sent to the UDS periodically but with an infrequent time period of, for example, every 6 to 12 hours. The data summary provides resiliency such that if any of the current data samples are lost in any of the various transfers, the data summary is still available. Local resiliency may be provided by storing a backlog of both current data sample files and summary data files in hard drive 421. For example, the four most recent current data sample files and the two most recent summary data files corresponding to each string name may be stored.

If FTP client 412b cannot send data from hard drive 421 to file system 425 for a predetermined period of time, for example, 15 minutes, the FTP client may notify the UDS and the UDS may notify each UDML. Each UDML then continues to cause the device driver to gather current statistical management data samples and add them to the data summaries at the same periodic interval (i.e., current data interval, e.g., 15 minutes), however, the UDML stops sending the current data samples to the UDS. Instead, the UDML sends only the data summaries to the UDS but at the more frequent current data interval (e.g., 15 minutes) instead of the longer time period (e.g., 6 to 12 hours). The UDS may then update the data summaries stored in hard drive 421 and cease collecting and storing current data samples. This will save space in the hard drive and minimize any data loss.

To reduce the amount of statistical management data being transferred to the UDS, a network manager may selectively configure only certain of the applications (e.g., device drivers) and certain of the interfaces to provide this data. As each UDML

registers with the UDS, the UDS may then inform each UDML with respect to each interface as to whether or not statistical management data should be gathered and sent to the UDS. There may be many circumstances in which gathering this data is unnecessary. For example, each ATM device driver may manage multiple virtual interfaces (VATMs) and within each VATM there may be several virtual circuits. A network manager may choose not to receive statistics for virtual circuits on which a customer has ordered only Variable Bit Rate (VBR) real time (VBR-rt) and VBR non-real time (VBR-nrt) service. For VBR-rt and VBR-nrt, the network service provider may provide the customer only with available / extra bandwidth and charge a simple flat fee per month. However, a network manager may need to receive statistics for virtual circuits on which a customer has ordered a high quality of service such as Constant Bit Rate (CBR) to ensure that the customer is getting the appropriate level of service and to appropriately charge the customer. In addition, a network manager may want to receive statistics for virtual circuits on which a customer has ordered Unspecified Bit Rate (UBR) service to police the customer's usage and ensure they are not receiving more network bandwidth than what they are paying for. Allowing a network manager to indicate that certain applications or certain interfaces managed by an application (e.g., a VATM) need not provide statistical management data or some portion of that data to the UDS reduces the amount of data transferred to the UDS – that is, reduces internal bandwidth utilization –, reduces the amount of storage space required in the hard drive, and reduces the processing power required to transfer the statistical management data from remote cards to external file system 425.

For each binary data file, the UDS creates a data summary file (e.g., data summary files 414a-414n) and stores it in, for example, hard drive 421. The data summary file defines the binary file format, including the type based on the string name, the length, the number of records and the version number. The UDS does not need to understand the binary data sent to it by each of the device drivers. The UDS need only combine data corresponding to similar string names into the same file and create a summary file based on the string name and the amount of data in the binary data file. The version number is passed to the UDS by the device driver, and the UDS includes the version number in the data summary file.

Periodically, FTP client 412b asynchronously reads each binary data file and corresponding data summary file from hard drive 421. Preferably, the FTP client reads these files from the hard drive through an out-of-band Ethernet connection, for example, Ethernet 32 (Fig. 1). Alternatively, the FTP client may read these files through an in-band data path 34 (Fig. 1). The FTP client then uses an FTP push to send the binary data file to a file system 425 accessible by the data collector server and, preferably local to the data collector server. The FTP client then uses another FTP push to send the data summary file to the local file system. Since binary data files may be very long and an FTP push of a binary data file may take some time, the data collector server may periodically search the local file system for data summary files. The data collector server may then attempt to open a discovered data summary file. If the data collector server is able to open the file, then that indicates that the FTP push of the data summary file is complete, and since the data summary file is pushed after the binary data file, the data collector server's ability to open the data summary file may be used as an indication that a new binary data file has been completely received. Since data summary files are much smaller than binary data files, having the data collector server look for and attempt to open data summary files instead of binary data files minimizes the thread wait within the data collector server.

In one embodiment, the data collector server is a JAVA program, and each different type of binary data file has a corresponding JAVA class file (e.g., class file 410a) that defines how the data collector server should process the binary data file. When a device driver is loaded into the network device, a corresponding JAVA class file is also loaded and stored in hard drive 421. The FTP client periodically polls the hard drive for new JAVA class files and uses an FTP push to send them to file system 425. The data collector server uses the binary file type in the data summary file to determine which JAVA class file it should use to interpret the binary data file. The data collector server then converts the binary data into ASCII or AMA/BAF format and stores the ASCII or AMA/BAF files in the file system. The data collector server may use a set of worker threads for concurrency.

As described, the data collector server is completely independent of and asynchronous with the FTP client, which is also independent and asynchronous of the UDS. The separation of the data collector server and FTP client avoids data loss due to process

synchronization problems, since there is no synchronization, and reduces the burden on the network device by not requiring the network device to maintain synchronization between the processes. In addition, if the data collector server goes down or is busy for some time, the FTP client and UDS continue working and continue sending binary data files and data summary files to the file system. When the data collector server is again available, it simply accesses the data summary files and processes the binary files as described above. Thus, there is no data loss and the limited storage capacity within the network device is not strained by storing data until the data collector server is available. In addition, if the FTP client or UDS goes down, the data collector server may continue working.

An NMS server (e.g., NMS server 851a), which may or may not be executing on the same computer system 62 as the data collector server, may periodically retrieve the ASCII or AMA/BAF files from the file system. The files may represent accounting, statistics, security, logging and/or other types of data gathered from hardware within the network device. The NMS server may also access the corresponding class files from the file system to learn how the data should be presented to a user, for example, how a graphical user interface (GUI) should be displayed, what data and format to display, or perhaps which one of many GUIs should be used. The NMS server may use the data to, for example, monitor network device performance, including quality of service guarantees and service level agreements, as well as bill customers for network usage. Alternatively, a separate billing server 423a or statistics server 423b, which may or may not be executing on the same computer system 62 as the data collector server and/or the NMS server, may periodically retrieve the ASCII or AMA/BAF files from the file system in order to monitor network device performance, including quality of service guarantees and service level agreements, and/or bill customers for network usage. One or more of the data collector server, the NMS server, the billing server and the statistics server may be combined into one server. Moreover, management files created by the data collector server may be combined with data from the configuration or NMS databases to generate billing records for each of the network provider's customers.

The data collector server may convert the ASCII or AMA/BAF files into other data formats, for example, Excel spread sheets, for use by the NMS server, billing server

and/or statistics server. In addition, the application class file for each data type may be modified to go beyond conversion, including direct integration into a database or an OSS system. For example, many OSS systems use a Portal billing system available from Portal Software, Inc. in Cupertino, CA. The JAVA class file associated with a particular binary data file and data summary file may cause the data collector server to convert the binary data file into ASCII data and then issue a Portal API call to give the ASCII data directly to the Portal billing system. As a result, accounting, statistics, logging and/or security data may be directly integrated into any other process, including third party processes, through JAVA class files.

Through JAVA class files, new device drivers may be added to a network device without having to change UDS 412a or FTP client 412b and without having to re-boot the network device and without having to upgrade/modify external processes. For example, a new forwarding card (e.g., forwarding card 552a) may be added to an operating network device and this new forwarding card may support MPLS. An MPLS device driver 419, linked within the UDML, is downloaded to the network device as well as a corresponding class file (e.g., class file 410e). When the FTP client discovers the new class file in hard drive 421, it uses an FTP push to send it to file system 425. The FTP client does not need to understand the data within the class file it simply needs to push it to the file system. Just as with other device drivers, the UDML causes the MPLS driver to register appropriate string names with the UDS and poll and send data to the UDS with a registered string name. The UDS stores binary data files (e.g., binary data file 416e) and corresponding data summary files (e.g., data summary file 414e) in the hard drive without having to understand the data within the binary data file. The FTP client then pushes these files to the file system again without having to understand the data. When the data summary file is discovered by the data collector server, the data collector server uses the binary file type in the data summary file to locate the new MPLS class file 410e in the file system and then uses the class file to convert the binary data in the corresponding binary data file into ASCII format and perhaps other data formats. Thus, a new device driver is added and statistical information may be gathered without having to change any of the other software and without having to re-boot the network device.

As described, having the data collector server be completely independent of and asynchronous with the FTP client avoids the typical problems encountered when internal and external management programs are synchronized. Moreover, modularity of device drivers and internal management programs is maintained by providing metadata through class files that instruct the external management programs as to how the management data should be processed. Consequently, device drivers may be modified, upgraded and added to an operating network device without disrupting the operation of any of the other device drivers or the management programs.

Configuration:

As described above, unlike a monolithic software architecture which is directly linked to the hardware of the computer system on which it runs, a modular software architecture includes independent applications that are significantly decoupled from the hardware through the use of a logical model of the computer system. Using the logical model and a code generation system, a view id and API are generated for each application to define each application's access to particular data in a configuration database and programming interfaces between the different applications. The configuration database is established using a data definition language (DDL) file also generated by the code generation system from the logical model. As a result, there is only a limited connection between the computer system's software and hardware, which allows for multiple versions of the same application to run on the computer system simultaneously and different types of applications to run simultaneously on the computer system. In addition, while the computer system is running, application upgrades and downgrades may be executed without affecting other applications and new hardware and software may be added to the system also without affecting other applications.

Referring again to Fig. 13b, initially, NMS 60 reads card table 47 and port table 49 to determine what hardware is available in computer system 10. The NMS assigns a logical identification number (LID) 98 (Figs. 14b and 14c) to each card and port and inserts these numbers in an LID to PID Card table (LPCT) 100 and an LID to PID Port table (LPPT) 101 in configuration database 42. Alternatively, the NMS could use the PID previously assigned to each board by the MCD. However, to allow for hardware redundancy, the NMS assigns an LID and may associate the LID with at

least two PIDs, a primary PID 102 and a backup PID 104. (LPCT 100 may include multiple backup PID fields to allow more than one backup PID to be assigned to each primary PID.)

The user chooses the desired redundancy structure and instructs the NMS as to which boards are primary boards and which boards are backup boards. For example, the NMS may assign LID 30 to line card 16a -- previously assigned PID 500 by the MCD -- as a user defined primary card, and the NMS may assign LID 30 to line card 16n – previously assigned PID 513 by the MCD – as a user defined back-up card (see row 106, Fig. 14b). The NMS may also assign LID 40 to port 44a -- previously assigned PID 1500 by the MCD -- as a primary port, and the NMS may assign LID 40 to port 68a – previously assigned PID 1600 by the MCD – as a back-up port (see row 107, Fig. 14c).

In a 1:1 redundant system, each backup line card backs-up only one other line card and the NMS assigns a unique primary PID and a unique backup PID to each LID (no LIDs share the same PIDs). In a 1:N redundant system, each backup line card backs-up at least two other line cards and the NMS assigns a different primary PID to each LID and the same backup PID to at least two LIDs. For example, if computer system 10 is a 1:N redundant system, then one line card, for example, line card 16n, serves as the hardware backup card for at least two other line cards, for example, line cards 16a and 16b. If the NMS assigns an LID of 31 to line card 16b, then in logical to physical card table 100 (see row 109, Fig. 14b), the NMS associates LID 31 with primary PID 501 (line card 16b) and backup PID 513 (line card 16n). As a result, backup PID 513 (line card 16n) is associated with both LID 30 and 31.

The logical to physical card table provides the user with maximum flexibility in choosing a redundancy structure. In the same computer system, the user may provide full redundancy (1:1), partial redundancy (1:N), no redundancy or a combination of these redundancy structures. For example, a network manager (user) may have certain customers that are willing to pay more to ensure their network availability, and the user may provide a backup line card for each of that customer's primary line cards (1:1). Other customers may be willing to pay for some redundancy but not full redundancy, and the user may provide one backup line card for all of that customer's

primary line cards (1:N). Still other customers may not need any redundancy, and the user will not provide any backup line cards for that customer's primary line cards. For no redundancy, the NMS would leave the backup PID field in the logical to physical table blank. Each of these customers may be serviced by separate computer systems or the same computer system. Redundancy is discussed in more detail below.

The NMS and MCD use the same numbering space for LIDs, PIDs and other assigned numbers to ensure that the numbers are different (no collisions).

The configuration database, for example, a Polyhedra relational database, supports an "active query" feature. Through the active query feature, other software applications can be notified of changes to configuration database records in which they are interested. The NMS database establishes an active query for all configuration database records to insure it is updated with all changes. The master SRM establishes an active query with configuration database 42 for LPCT 100 and LPPT 101. Consequently, when the NMS adds to or changes these tables, configuration database 42 sends a notification to the master SRM and includes the change. In this example, configuration database 42 notifies master SRM 36 that LID 30 has been assigned to PID 500 and 513 and LID 31 has been assigned to PID 501 and 513. The master SRM then uses card table 47 to determine the physical location of boards associated with new or changed LIDs and then tells the corresponding slave SRM of its assigned LID(s). In the continuing example, master SRM reads CT 47 to learn that PID 500 is line card 16a, PID 501 is line card 16b and PID 513 is line card 16n. The master SRM then notifies slave SRM 37b on line card 16a that it has been assigned LID 30 and is a primary line card, SRM 37c on line card 16b that it has been assigned LID 31 and is a primary line card and SRM 37o on line card 16n that it has been assigned LIDs 30 and 31 and is a backup line card. All three slave SRMs 37b, 37c and 37o then set up active queries with configuration database 42 to insure that they are notified of any software load records (SLRs) created for their LIDs. A similar process is followed for the LIDs assigned to each port.

The NMS informs the user of the hardware available in computer system 10. This information may be provided as a text list, as a logical picture in a graphical user

interface (GUI), or in a variety of other formats. The user then uses the GUI to tell the NMS (e.g., NMS client 850a, Fig. 2a) how they want the system configured.

The user will select which ports (e.g., 44a-44d, 46a-46f, 68a-68n) the NMS should enable. There may be instances where some ports are not currently needed and, therefore, not enabled. The user also needs to provide the NMS with information about the type of network connection (e.g., connection 70a-70d, 72a-72f, 74a-74n). For example, the user may want all ports 44a-44d on line card 16a enabled to run ATM over SONET. The NMS may start one ATM application to control all four ports, or, for resiliency, the NMS may start one ATM application for each port. Alternatively, each port may be enabled to run a different protocol (e.g., MPLS, IP, Frame Relay).

In the example given above, the user must also indicate the type of SONET fiber they have connected to each port and what paths to expect. For example, the user may indicate that each port 44a-44d is connected to a SONET optical fiber carrying an OC-48 stream. A channelized OC-48 stream is capable of carrying forty-eight STS-1 paths, sixteen STS-3c paths, four STS-12c paths or a combination of STS-1, STS-3c and STS-12c paths. A clear channel OC-48c stream carries one concatenated STS-48 path. In the example, the user may indicate that the network connection to port 44a is a clear channel OC-48 SONET stream having one STS-48 path, the network connection to port 44b is a channelized OC-48 SONET stream having three STS-12c paths (i.e., the SONET fiber is not at full capacity – more paths may be added later), the network connection to port 44c is a channelized OC-48 SONET stream having two STS-3c paths (not at full capacity) and the network connection to port 44d is a channelized OC-48 SONET stream having three STS-12c paths (not at full capacity). In the current example, all paths within each stream carry data transmitted according to the ATM protocol. Alternatively, each path within a stream may carry data transmitted according to a different protocol.

The NMS (e.g., NMS server 851a-851n) uses the information received from the user (through the GUI / NMS client) to create records in several tables in the configuration database, which are then copied to the NMS database. These tables are accessed by other applications to configure computer system 10. One table, the service endpoint

table (SET) 76 (see also Fig. 14a), is created when the NMS assigns a unique service endpoint number (SE) to each path on each enabled port and corresponds each service endpoint number with the physical identification number (PID) previously assigned to each port by the MCD. Through the use of the logical to physical port table (LPPT), the service endpoint number also corresponds to the logical identification number (LID) of the port. For example, since the user indicated that port 44a (PID 1500) has a single STS-48 path, the NMS assigns one service endpoint number (e.g. SE 1, see row 78, Fig. 14a). Similarly, the NMS assigns three service endpoint numbers (e.g., SE 2, 3, 4, see rows 80-84) to port 44b (PID 1501), two service endpoint numbers (e.g., SE 5, 6, see rows 86, 88) to port 44c (PID 1502) and three service endpoint numbers (e.g., SE 7, 8, 9, see rows 90, 92, 94) to port 44d.

Service endpoint managers (SEMs) within the modular system services of the kernel software running on each line card use the service endpoint numbers assigned by the NMS to enable ports and to link instances of applications, for example, ATM, running on the line cards with the correct port. The kernel may start one SEM to handle all ports on one line card, or, for resiliency, the kernel may start one SEM for each particular port. For example, SEMs 96a-96d are spawned to independently control ports 44a-44d.

The service endpoint managers (SEMs) running on each board establish active queries with the configuration database for SET 76. Thus, when the NMS changes or adds to the service endpoint table (SET), the configuration database sends the service endpoint manager associated with the port PID in the SET a change notification including information on the change that was made. In the continuing example, configuration database 42 notifies SEM 96a that SET 76 has been changed and that SE 1 was assigned to port 44a (PID 1500). Configuration database 42 notifies SEM 96b that SE 2, 3, and 4 were assigned to port 44b (PID 1501), SEM 96c that SE 5 and 6 were assigned to port 44c (PID 1502) and SEM 96d that SE 7, 8, and 9 were assigned to port 44d (PID 1503). When a service endpoint is assigned to a port, the SEM associated with that port passes the assigned SE number to the port driver for that port using the port PID number associated with the SE number.

To load instances of software applications on the correct boards, the NMS creates software load records (SLR) 128a-128n in configuration database 42. The SLR includes the name 130 (Fig. 14f) of a control shim executable file and an LID 132 for cards on which the application must be spawned. In the continuing example, NMS 60 creates SLR 128a including the executable name atm_cntrl.exe and card LID 30 (row 134). The configuration database detects LID 30 in SLR 128a and sends slave SRMs 37b (line card 16a) and 37o (line card 16n) a change notification including the name of the executable file (e.g., atm_cntrl.exe) to be loaded. The primary slave SRMs then download and execute a copy of atm_cntrl.exe 135 from memory 40 to spawn the ATM controllers (e.g., ATM controller 136 on line card 16a). Since slave SRM 37o is on backup line card 16n, it may or may not spawn an ATM controller in backup mode. Software backup is described in more detail below. Instead of downloading a copy of atm_cntrl.exe 135 from memory 40, a slave SRM may download it from another line card that already downloaded a copy from memory 40. There may be instances when downloading from a line card is quicker than downloading from central processor 12. Through software load records and the tables in configuration database 42, applications are downloaded and executed without the need for the system services, including the SRM, or any other software in the kernel to have information as to how the applications should be configured. The control shims (e.g., atm_cntrl.exe 135) interpret the next layer of the application (e.g., ATM) configuration.

For each application that needs to be spawned, for example, an ATM application and a SONET application, the NMS creates an application group table. Referring to Fig. 14d, ATM group table 108 indicates that four instances of ATM (i.e., group number 1, 2, 3, 4) – corresponding to four enabled ports 44a-44n -- are to be started on line card 16a (LID 30). If other instances of ATM are started on other line cards, they would also be listed in ATM group table 108 but associated with the appropriate line card LID. ATM group table 108 may also include additional information needed to execute ATM applications on each particular line card. (See description of software backup below.)

In the above example, one instance of ATM was started for each port on the line card. This provides resiliency and fault isolation should one instance of ATM fail or should

one port suffer a failure. An even more resilient scheme would include multiple instances of ATM for each port. For example, one instance of ATM may be started for each path received by a port.

The application controllers on each board now need to know how many instances of the corresponding application they need to spawn. This information is in the application group table in the configuration database. Through the active query feature, the configuration database notifies the application controller of records associated with the board's LID from corresponding application group tables. In the continuing example, configuration database 42 sends ATM controller 136 records from ATM group table 108 that correspond to LID 30 (line card 16a). With these records, ATM controller 136 learns that there are four ATM groups associated with LID 30 meaning ATM must be instantiated four times on line card 16a. ATM controller 136 asks slave SRM 37b to download and execute four instances (ATM 110-113, Fig. 15) of atm.exe 138.

Once spawned, each instantiation of ATM 110-113 sends an active database query to search ATM interface table 114 for its corresponding group number and to retrieve associated records. The data in the records indicates how many ATM interfaces each instantiation of ATM needs to spawn. Alternatively, a master ATM application (not shown) running on central processor 12 may perform active queries of the configuration database and pass information to each slave ATM application running on the various line cards regarding the number of ATM interfaces each slave ATM application needs to spawn.

Referring to Figs. 14e and 15, for each instance of ATM 110-113 there may be one or more ATM interfaces. To configure these ATM interfaces, the NMS creates an ATM interface table 114. There may be one ATM interface 115-122 per path / service endpoint or multiple virtual ATM interfaces 123-125 per path. This flexibility is left up to the user and NMS, and the ATM interface table allows the NMS to communicate this configuration information to each instance of each application running on the different line cards. For example, ATM interface table 114 indicates that for ATM group 1, service endpoint 1, there are three virtual ATM interfaces

(ATM-IF 1-3) and for ATM group 2, there is one ATM interface for each service endpoint: ATM-IF 4 and SE 2; ATM-IF 5 and SE 3; and ATM-IF 6 and SE 4.

Computer system 10 is now ready to operate as a network switch using line card 16a and ports 44a-44d. The user will likely provide the NMS with further instructions to configure more of computer system 10. For example, instances of other software applications, such as an IP application, and additional instances of ATM may be spawned (as described above) on line cards 16a or other boards in computer system 10.

As shown above, all application dependent data resides in memory 40 and not in kernel software. Consequently, changes may be made to applications and configuration data in memory 40 to allow hot (while computer system 10 is running) upgrades of software and hardware and configuration changes. Although the above described power-up and configuration of computer system 10 is complex, it provides massive flexibility as described in more detail below.

Template Driven Service Provisioning:

Instead of using the GUI to interactively provision services on one network device in real time, a user may provision services on one or more network devices in one or more networks controlled by one or more network management systems (NMSs) interactively and non-interactively using an Operations Support Services (OSS) client and templates. At the heart of any carrier's network is the OSS, which provides the overall network management infrastructure and the main user interface for network managers / administrators. The OSS is responsible for consolidating a diverse set of element/network management systems and third-party applications into a single system that is used, for example, to detect and resolve network faults (**F**ault Management), configure and upgrade the network (**C**onfiguration Management), account and bill for network usage (**A**ccounting Management), oversee and tune network performance (**P**erformance Management), and ensure ironclad network security (**S**ecurity Management). FCAPS are the five functional areas of network management as defined by the International Organization for Standardization (ISO). Through templates one or more NMSs may be integrated with a telecommunication network carrier's OSS.

Templates are metadata and include scripts of instructions and parameters. In one embodiment, instructions within templates are written in ASCII text to be human readable. There are three general categories of templates, provisioning templates, control templates and batch templates. A user may interactively connect the OSS client with a particular NMS server and then cause the NMS server to connect to a particular device. Instead, the user may create a control template that non-interactively establishes these connections. Once the connections are established, whether interactively or non-interactively, provisioning templates may be used to complete particular provisioning tasks. The instructions within a provisioning template cause the OSS client to issue appropriate calls to the NMS server which cause the NMS server to complete the provisioning task, for example, by writing / modifying data within the network device's configuration database. Batch templates may be used to concatenate a series of templates and template modifications (i.e., one or more control and provisioning templates) to provision one or more network devices. Through the client / server based architecture, multiple OSS clients may work with one or more NMS servers. Database view ids and APIs for the OSS client may be generated using the logical model and code generation system (Fig. 3b) to synchronize the integration interfaces between the OSS clients and the NMS servers.

Interactively, a network manager may have an OSS client execute many provisioning templates to complete many provisioning tasks. Instead, the network manager may order and sequence the execution of many provisioning templates within a batch template to non-interactively complete the many provisioning tasks and build custom services. In addition, execution commands followed by control template names may be included within batch templates to non-interactively cause an OSS client to establish connections with particular NMS servers and network devices. For example, a first control template may designate a network device to which the current OSS client and NMS server are not connected. Including an execution command followed by the first control template name in a batch template will cause the OSS client to issue calls to the NMS server to cause the NMS server to access the different network device. As another example, a second control template may designate an NMS server and a network device to which the OSS client is not currently connected. Including an execution command followed by the second control template name will

cause the OSS client to set up connections to both the different NMS server and the different network device. Moreover, batch templates may include execution commands followed by provisioning template names after each execution command and control template to provision services within the network devices designated by the control templates. Through batch templates, therefore, multiple control templates and provisioning templates may be ordered and sequenced to provision services within multiple network devices in multiple networks controlled by multiple NMSs.

Calls issued by the OSS client to the NMS server may cause the NMS server to immediately provision services or delay provisioning services until a predetermined time, for example, a time when the network device is less likely to be busy. Templates may be written to apply to different types of network devices.

A "command line" interactive interpreter within the OSS client may be used by a network manager to select and modify existing templates or to create new templates. Templates may be generated for many various provisioning tasks, for example, setting up a permanent virtual circuit (PVC), a switched virtual circuit (SVC), a SONET path (SPATH), a traffic descriptor (TD) or a virtual ATM interface (VAIF). Once a template is created, a network manager change default parameters within the template to complete particular provisioning tasks. A network manager may also copy a template and modify it to create a new template.

Referring to Fig. 3h, using the interactive interpreter, a network administrator may provision services by selecting (step 888) a template and using the default parameters within that template or copying and renaming (step 889) a particular provisioning template corresponding to a particular provisioning task and either accepting default parameter values provided by the template or changing (step 890) those default values to meet the administrator's needs. The network administrator may also change parameters and instructions within a copy of a template to create a new template. The modified provisioning templates are sent to or loaded into (step 891) the OSS client, which executes the instructions within the template and issues the appropriate calls (step 892) to the NMS server to satisfy the provisioning need. The OSS client may be written in JAVA and employ script technology. In response to calls received from the OSS client, the NMS server may execute (step 894) the provisioning requests defined

by a template immediately or in a "batch-mode" (step 893), perhaps with other calls received from the OSS client or other clients, at a time when network transactions are typically low (e.g., late at night).

Referring to Fig. 3i, at the interactive interpreter prompt 912 (e.g., Enetcli>) a network manager may type in "help" and be provided with a list (e.g., list 913) of commands that are available. In one embodiment, available commands may include bye, close, execute, help, load, manage, open, quit, showCurrent, showTemplate, set, status, writeCurrent, and writeTemplate. Many different commands are possible. The bye command allows the network manager to exit the interactive interpreter, the close command allows the network manager to close a connection between the OSS client and that NMS server, and the execute command followed by a template type causes the OSS client to execute the instructions within the loaded template corresponding to that template type.

As shown, the help command alone causes the interactive interpreter to display the list of commands. The help command followed by another command provides help information about that command. The load command followed by a template type and a named template loads the named template into the OSS client such that any commands followed by the template type will use the named/loaded template. The manage command followed by an IP address of a network device causes the OSS client to issue a call to an NMS server to establish a connection between the NMS server and that network device. Alternatively, a username and password may also need to be supplied. The open command followed by an NMS server IP address causes the OSS client to open a connection with that NMS server, and again, the network manager may also need to supply a username and password. Instead of an IP address, a domain name server (DNS) name may be provided and a host look up may be used to determine the IP address and access the corresponding device.

The showCurrent command followed by a template type will cause the interactive interpreter to display current parameter values for the loaded template corresponding to that template type. For example, showCurrent SPATH 914 displays a list 915 of parameters and current parameter values for the loaded template corresponding to the SPATH template type. The showTemplate command followed by a template type

will cause the OSS client to display available parameters and acceptable parameter values for each parameter within the loaded template. For example, showTemplate SPATH 916 causes the interactive interpreter to display the available parameters 917 within the loaded template corresponding to the SPATH template type. The set command followed by a template type, a parameter name and a value will change the named parameter to the designated value within the loaded template, and a subsequent showCurrent command followed by that template type will show the new parameter value within the loaded.

The status command 918 will cause the interactive interpreter to display a status of the current interactive interpreter session. For example, the interactive interpreter may display the name 919 of an NMS server to which the OSS client is currently connected (as shown in Fig. 3i, the OSS client is currently not connected to an NMS server) and the interactive interpreter may display the names 920 of available template types. The writeCurrent command followed by a template type and a new template name will cause the interactive interpreter to make a copy of the loaded template, including current parameter values, with the new template name. The writeTemplate command followed by a template type and a new template name, will cause the interactive interpreter to make a copy of the template with the new template name with placeholders values (i.e., <String>) that indicate the network manager needs to fill in the template with the required datatypes as parameter values. The network manager may then use the load command followed by the new template name to load the new template into the OSS client.

Referring to Fig. 3j, from the interactive interpreter prompt (e.g., Enetcli>), a network manager may interactively provision services on a network device. The network manager begins by typing an open command 921a followed by the IP address of an NMS server to cause the OSS client to open a connection 921b with that NMS server. The network manager may then issue a manage command 921c followed by the IP address of a particular network device to cause the OSS client to issue a call 921d to the NMS server to cause the NMS server to open a connection 921e with that network device.

The network manager may now provision services within that network device by typing in an execute command 921f followed by a template type. For example, the network manager may type "execute SPATH" at the Enetcli> prompt to cause the OSS client to execute the instructions 921g within the loaded SPATH template using the parameter values within the loaded SPATH template. Executing the instructions causes the OSS client to issue calls to the NMS server, and these calls cause the NMS server to complete the provisioning task 921h. For example, following an execute SPATH command, the NMS server will set up a SONET path in the network device using the parameter values passed to the NMS server by the OSS client from the template.

At any time from the Enetcli> prompt, a network manager may change the parameter values within a template. Again, the network manager may use showCurrent followed by a template type to see the current parameter values within the loaded template or showTemplate to see the available parameters within the loaded template. The network manager may then use the set command followed by the template type, parameter name and new parameter value to change a parameter value within the loaded template. For example, after the network manager sets up a SONET path within the network device, the network manager may change one or more parameter values within the loaded SPATH template and re-execute the SPATH template to set up a different SONET path within the same network device.

Once a connection to a network device is open, the network manager may interactively execute any template any number of times to provision services within that network device. The network manager may also create new templates and execute those. The network manager may simply write a new template or use the writeCurrent or writeTemplate commands to copy an existing template into a new template name and then edit the instructions within the new template.

After provisioning services within a first network device, the network manager may open a connection with a second network device to provision services within that second network device. If the NMS server currently connected to the OSS client is capable of establishing a connection with the second network device, then the network manager may simply open a connection to the second network device. If the

NMS server currently connected to the OSS client is not capable of establishing a connection with the second network device, then the network manager closes the connections with the NMS server and then opens connections with a second NMS server and the second network device. Thus, a network manager may easily manage / provision services within multiple network devices within multiple networks even if they are managed by different NMS servers. In addition, other network managers may provision services on the same network devices through the same NMS servers using other OSS clients that are perhaps running on other computer systems. That is, multiple OSS clients may be connected to multiple NMS servers.

Instead of interactively establishing connections with NMS servers and network devices, control templates may be used to non-interactively establish these connections. Referring to Fig. 3k, using a showCurrent command 922 followed by CONTROL causes the interactive interpreter to display parameters available in the loaded CONTROL template. In one embodiment, an execute control command will automatically cause the OSS client to execute instructions within the loaded CONTROL template and open a connection to an NMS server designated within the CONTROL template. Since the OSS client automatically opens a connection with the designated NMS server, the open command may but need not be included within the CONTROL template. In this example, the CONTROL template includes "localhost" 923a as the DNS name of the NMS server with which the OSS client should open a connection. In one embodiment, "localhost" refers to the same system as the OSS client. A username 923b and password 923c may also need to be used to open the connection with the localhost NMS server. The CONTROL template also includes the manage command 923d and a network device IP address 923e of 192.168.9.202. With this information (and perhaps the username and password or another username and password), the OSS client issues calls to the localhost NMS server to cause the server to set up a connection with that network device.

The template may also include an output file name 923f where any output / status information generated in response to the execution of the CONTROL template will be sent. The template may also include a version number 923g. Version numbers allow a new template to be created with the same name as an old template but with a new version number, and the new template may include additional/different parameters

and/or instructions. Using version numbers, both old (e.g., not upgraded) and new OSS clients may use the templates but only access those templates having particular version numbers that correspond to the functionality of each OSS client.

Once connections with an NMS server and network device are established (either interactively or non-interactively through a control template), services within the network device may be provisioned. As described above, a network manager may interactively provision services by issuing execute commands followed by provisioning template types. Alternatively, a network manager may provision services non-interactively through batch templates, which include an ordered list of tasks, including execute commands followed by provisioning template types.

Referring to Fig. 3L, a batch template type named BATCH 924 includes an ordered list of tasks, including execute commands followed by provisioning template types. When a network manager issues an execute command followed by the BATCH template type at the Enetcli> prompt, the OSS client will carry out each of the tasks within the loaded BATCH template. In this example, task1 924a includes "execute SPATH" which causes the OSS client to establish a SONET path within the network device to which a connection is open, task2 924b includes "execute PVC" to cause the OSS client to set up a permanent virtual circuit within the network device, and task3 924c includes "execute SPVC" to cause the OSS client to set up a soft permanent virtual circuit within the network device.

If multiple similar provisioning tasks are needed, then the network manager may use writeCurrent or writeTemplate to create multiple similar templates (i.e., same template type with different template names), change or add parameter values within these multiple similar templates using the set command, and sequentially load and execute each of the different named templates. For example, SPVC is the template type and task3 causes the OSS to execute instructions within the previously loaded named template. Spvc1 and spvc2 are two different named templates (or template instantiations) corresponding to the SPVC template type for setting up soft permanent virtual circuits having different parameters from each other and the loaded template to set up different SPVCs. In this example, the BATCH template then includes task4 924d including "load SPVC spvc1" to load the spvc1 template and then task5 924e

"execute SPVC" to cause the OSS client to execute the loaded spvc1 template and set up a different SPVC. Similarly, task6 924f includes "load SPVC spvc2" and task7 924e includes "execute SPVC" to cause the OSS client to execute the loaded spvc2 template and set up yet another different SPVC.

Alternatively, the batch template may include commands for altering an existing template such that multiple similar templates are not necessary. For example, the loaded BATCH template may include task50 924g "set SPATH PortID 3" to cause the OSS client to change the PortID parameter within the SPATH template to 3. The BATCH template then includes task51 924h "execute SPATH" 924g to cause the OSS client to execute the SPATH template including the new parameter value which sets up a different SONET path. A BATCH template may include many set commands to change parameter values followed by execute commands to provision multiple similar services within the same network device. For example, the BATCH template may further include task52 924i "set SPATH SlotID 2" followed by task53 924j "execute SPATH" to set up yet another different SONET path. Using this combination of set and execute commands eliminates the need to write, store and keep track of multiple similar templates.

Batch templates may also be used to non-interactively provision services within multiple different network devices by ordering and sequencing tasks including execute commands followed by control template types and then execute commands followed by provisioning template types. Referring to Fig. 3M, instead of non-interactively establishing connections with an NMS server and a network device using a control template, a batch template may be used. For example, the first task in a loaded BATCH template 925 may be task1 925a "execute CONTROL". This will cause the OSS client to execute the loaded CONTROL template to establish connections with the NMS server and the network device designated within the loaded CONTROL template (e.g., localhost and 192.168.9.202). The BATCH template then includes provisioning tasks, for example, task2 925b includes "execute SPATH" to set up a SONET path, and task3 925c includes "set SPATH PortID 3" and task4 925d includes "execute SPATH" to set up a different SONET path. Many additional provisioning tasks for this network device may be completed in this way.

The BATCH template may then have a task including a set command to modify one or more parameters within a control template to cause the OSS client to set up a connection with a different network device and perhaps a different NMS server. Where the network manager wishes to provision a network device capable of being connected to through the currently connected NMS server, for example, localhost, then the BATCH template need only have task61 925e including "set CONTROL System" followed by the IP address of the different network device, for example, 192.168.9.201. The BATCH template then has a task62 925f including "execute CONTROL", which causes the OSS client to issue calls to the localhost NMS server to establish a connection with the different network device. The BATCH template may then have tasks including execute commands followed by provisioning templates, for example, task63 925g including "execute SPATH", to provision services within the different network device.

If the network manager wishes to provision a network device coupled with another NMS server, then the BATCH template includes, for example, task108 925h including "close" to drop the connection between the OSS client and localhost NMS server. The BATCH template may then have, for example, task109 925i including "set CONTROL Server Server1" to change the server parameter within the loaded CONTROL template to Server1 and task110 925j including "set CONTROL System 192.168.8.200" to change the network device parameter within the loaded CONTROL template to the IP address of the new network device. The BATCH template may then have task111 925k including "execute CONTROL" to cause the OSS client to set up connections to the Server1 NMS server and to network device 192.168.8.200. The BATCH template may then include tasks with execute commands followed by provisioning template types to provision services within the network device, for example, task112 925L includes "execute SPATH".

The templates and interactive interpreter / OSS client may be loaded and executed on a central OSS computer system(s) and used to provision services in one or more network devices in one or more network domains. A network administrator may install an OSS client at various locations and/or for "manage anywhere" purposes, web technology may be used to allow a network manager to download an OSS client program from a web accessible server onto a computer at any location. The network

manager may then use the OSS client in the same manner as when it is loaded onto a central OSS computer system. Thus, the network manager may provision services from any computer at any location.

Provisioning templates may be written to apply to different types of network devices. The network administrator does not need to know details of the network device being provisioned as the parameters required and available for modification are listed in the various templates. Consequently, the templates allow for multifaceted integration of different network management systems (NMS) into existing OSS infrastructures.

Instead of using template executable files and an OSS client, network managers may prefer to use their standard OSS interface to provision services in various network devices. In one embodiment, therefore, a single OSS client application programming interface (API) and a library of compiled code may be linked directly into the OSS software. The library of compiled code is a subset of the compiled code used to create the OSS client, with built-in templates including provisioning, control, batch and other types of templates. The OSS software then uses the supported templates as documentation of the necessary parameters needed for each provisioning task and presents template streams (null terminated arrays of arguments that serialize the totality of arguments required to construct a supported template) via the single API for potential alteration through the OSS standard interface. Since the network managers are comfortable working with the OSS interface, provisioning services may be made more efficient and simple by directly linking the OSS client API and templates into the OSS software.

Typically, OSS software is written in C or C++ programming language. In one embodiment, the OSS client and templates are written in JAVA, and JAVA Native Interface (JNI) is used by the OSS software to access the JAVA OSS client API and templates.

Inter-Process Communication:

As described above, the operating system assigns a unique process identification number (proc_id) to each spawned process. Each process has a name, and each

process knows the names of other processes with which it needs to communicate. The operating system keeps a list of process names and the assigned process identification numbers. Processes send messages to other processes using the assigned process identification numbers without regard to what board is executing each process (i.e., process location). Application Programming Interfaces (APIs) define the format and type of information included in the messages.

The modular software architecture configuration model requires a single software process to support multiple configurable objects. For example, as described above, an ATM application may support configurations requiring multiple ATM interfaces and thousands of permanent virtual connections per ATM interface. The number of processes and configurable objects in a modular software architecture can quickly grow especially in a distributed processing system. If the operating system assigns a new process for each configurable object, the operating system's capabilities may be quickly exceeded.

For example, the operating system may be unable to assign a process for each ATM interface, each service endpoint, each permanent virtual circuit, etc.. In some instances, the process identification numbering scheme itself may not be large enough. Where protected memory is supported, the system may have insufficient memory to assign each process and configurable object a separate memory block. In addition, supporting a large number of independent processes may reduce the operating system's efficiency and slow the operation of the entire computer system.

One alternative is to assign a unique process identification number to only certain high level processes. Referring to Fig. 16a, for example, process identification numbers may only be assigned to each ATM process (e.g., ATMs 240, 241) and not to each ATM interface (e.g., ATM IFs 242-247) and process identification numbers may only be assigned to each port device driver (e.g., device drivers 248, 250, 252) and not to each service endpoint (e.g., SE 253-261). A disadvantage to this approach is that objects within one high level process will likely need to communicate with objects within other high level processes. For example, ATM interface 242 within ATM 240 may need to communicate with SE 253 within device driver 248. ATM IF 242 needs to know if SE 253 is active and perhaps certain other information about SE 253. Since SE 253 was not assigned a process identification number, however,

neither ATM 240 nor ATM IF 242 knows if it exists. Similarly, ATM IF 242 knows it needs to communicate with SE 253 but does not know that device driver 248 controls SE 253.

One possible solution is to hard code the name of device driver 248 into ATM 240. ATM 240 then knows it must communicate with device driver 248 to learn about the existence of any service endpoints within device driver 248 that may be needed by ATM IF 242, 243 or 244. Unfortunately, this can lead to scalability issues. For instance, each instantiation of ATM (e.g., ATM 240, 241) needs to know the name of all device drivers (e.g., device drivers 248, 250, 252) and must query each device driver to locate each needed service endpoint. An ATM query to a device driver that does not include a necessary service endpoint is a waste of time and resources. In addition, each high level process must periodically poll other high level processes to determine whether objects within them are still active (i.e., not terminated) and whether new objects have been started. If the object status has not changed between polls, then the poll wasted resources. If the status did change, then communications have been stalled for the length of time between polls. In addition, if a new device driver is added (e.g., device driver 262), then ATM 240 and 241 cannot communicate with it or any of the service endpoints within it until they have been upgraded to include the new device driver's name.

Preferably, computer system 10 implements a name server process and a flexible naming procedure. The name server process allows high level processes to register information about the objects within them and to subscribe for information about the objects with which they need to communicate. The flexible naming procedure is used instead of hard coding names in processes. Each process, for example, applications and device drivers, use tables in the configuration database to derive the names of other configurable objects with which they need to communicate. For example, both an ATM application and a device driver process may use an assigned service endpoint number from the service endpoint table (SET) to derive the name of the service endpoint that is registered by the device driver and subscribed for by the ATM application. Since the service endpoint numbers are assigned by the NMS during configuration, stored in SET 76 and passed to local SEMs, they will not be changed if device drivers or applications are upgraded or restarted.

Referring to Fig. 16b, for example, when device drivers 248, 250 and 252 are started they each register with name server (NS) 264. Each device driver provides a name, a process identification number and the name of each of its service endpoints. Each device driver also updates the name server as service endpoints are started, terminated or restarted. Similarly, each instantiation of ATM 240, 241 subscribes with name server 264 and provides its name, process identification number and the name of each of the service endpoints in which it is interested. The name server then notifies ATM 240 and 241 as to the process identification of the device driver with which they should communicate to reach a desired service endpoint. The name server updates ATM 240 and 241 in accordance with updates from the device drivers. As a result, updates are provided only when necessary (i.e., no wasted resources), and the computer system is highly scalable. For example, if a new device driver 262 is started, it simply registers with name server 264, and name server 264 notifies either ATM 240 or 241 if a service endpoint in which they are interested is within the new device driver. The same is true if a new instantiation of ATM – perhaps an upgraded version -- is started or if either an ATM application or a device driver fails and is restarted.

Referring to Fig. 16c, when the SEM, for example, SEM 96a, notifies a device driver, for example, device driver (DD) 222, of its assigned SE number, DD 222 uses the SE number to generate a device driver name. In the continuing example from above, where the ATM over SONET protocol is to be delivered to port 44a and DD 222, the device driver name may be for example, atm.se1. DD 222 publishes this name to NS 220b along with the process identification assigned by the operating system and the name of its service endpoints.

Applications, for example, ATM 224, also use SE numbers to generate the names of device drivers with which they need to communicate and subscribe to NS 220b for those device driver names, for example, atm.se1. If the device driver has published its name and process identification with NS 220b, then NS 220b notifies ATM 224 of the process identification number associated with atm.se1 and the name of its service endpoints. ATM 224 can then use the process identification to communicate with DD 222 and, hence, any objects within DD 222. If device driver 222 is restarted or

upgraded, SEM 96a will again notify DD 222 that its associated service endpoint is SE 1 which will cause DD 222 to generate the same name of atm.se1. DD 222 will then re-publish with NS 220b and include the newly assigned process identification number. NS 220b will provide the new process identification number to ATM 224 to allow the processes to continue to communicate. Similarly, if ATM 224 is restarted or upgraded, it will use the service endpoint numbers from ATM interface table 114 and, as a result, derive the same name of atm.se1 for DD 222. ATM 224 will then re-subscribe with NS 220b.

Computer system 10 includes a distributed name server (NS) application including a name server process 220a-220n on each board (central processor and line card). Each name server process handles the registration and subscription for the processes on its corresponding board. For distributed applications, after each application (e.g., ATM 224a-224n) registers with its local name server (e.g., 220b-220n), the name server registers the application with each of the other name servers. In this way, only distributed applications are registered / subscribed system wide which avoids wasting system resources by registering local processes system wide.

The operating system, through the use of assigned process identification numbers, allows for inter-process communication (IPC) regardless of the location of the processes within the computer system. The flexible naming process allows applications to use data in the configuration database to determine the names of other applications and configurable objects, thus, alleviating the need for hard coded process names. The name server notifies individual processes of the existence of the processes and objects with which they need to communicate and the process identification numbers needed for that communication. The termination, re-start or upgrade of an object or process is, therefore, transparent to other processes, with the exception of being notified of new process identification numbers. For example, due to a configuration change initiated by the user of the computer system, service endpoint 253 (Fig. 16b), may be terminated within device driver 248 and started instead within device driver 250. This movement of the location of object 253 is transparent to both ATM 240 and 241. Name server 264 simply notifies whichever processes have subscribed for SE 253 of the newly assigned process identification number corresponding to device driver 250.

The name server or a separate binding object manager (BOM) process may allow processes and configurable objects to pass additional information adding further flexibility to inter-process communications. For example, flexibility may be added to the application programming interfaces (APIs) used between processes. As discussed above, once a process is given a process identification number by the name server corresponding to an object with which it needs to communicate, the process can then send messages to the other process in accordance with a predefined application programming interface (API). Instead of having a predefined API, the API could have variables defined by data passed through the name server or BOM, and instead of having a single API, multiple APIs may be available and the selection of the API may be dependent upon information passed by the name server or BOM to the subscribed application.

Referring to Fig. 16d, a typical API will have a predefined message format 270 including, for example, a message type 272 and a value 274 of a fixed number of bits (e.g., 32). Processes that use this API must use the predefined message format. If a process is upgraded, it will be forced to use the same message format or change the API / message format which would require that all processes that use this API also be similarly upgraded to use the new API. Instead, the message format can be made more flexible by passing information through the name server or BOM. For example, instead of having the value field 274 be a fixed number of bits, when an application registers a name and process identification number it may also register the number of bits it plans on using for the value field (or any other field). Perhaps a zero indicates a value field of 32 bits and a one indicates a value filed of 64 bits. Thus, both processes know the message format but some flexibility has been added.

In addition to adding flexibility to the size of fields in a message format, flexibility may be added to the overall message format including the type of fields included in the message. When a process registers its name and process identification number, it may also register a version number indicating which API version should be used by other processes wishing to communicate with it. For example, device driver 250 (Fig. 16b) may register SE 258 with NS 264 and provide the name of SE 258, device driver 250's process identification number and a version number one, and device driver 252

may register SE 261 with NS 264 and provide the name of SE 261, device driver 252's process identification number and a version number (e.g., version number two). If ATM 240 has subscribed for either SE 258 or SE 261, then NS 264 notifies ATM 240 that SE 258 and SE 261 exist and provides the process identification numbers and version numbers. The version number tells ATM 240 what message format and information SE 258 and SE 261 expect. The different message formats for each version may be hard coded into ATM 240 or ATM 240 may access system memory or the configuration database for the message formats corresponding to service endpoint version one and version two. As a result, the same application may communicate with different versions of the same configurable object using a different API.

This also allows an application, for example, ATM, to be upgraded to support new configurable objects, for example, new ATM interfaces, while still being backward compatible by supporting older configurable objects, for example, old ATM interfaces. Backward compatibility has been provided in the past through revision numbers, however, initial communication between processes involved polling to determine version numbers and where multiple applications need to communicate, each would need to poll the other. The name server / BOM eliminates the need for polling.

As described above, the name server notifies subscriber applications each time a subscribed for process is terminated. Instead, the name server / BOM may not send such a notification unless the System Resiliency Manager (SRM) tells the name server / BOM to send such a notification. For example, depending upon the fault policy / resiliency of the system, a particular software fault may simply require that a process be restarted. In such a situation, the name server / BOM may not notify subscriber applications of the termination of the failed process and instead simply notify the subscriber applications of the newly assigned process identification number after the failed process has been restarted. Data that is sent by the subscriber processes after the termination of the failed process and prior to the notification of the new process identification number may be lost but the recovery of this data (if any) may be less problematic than notifying the subscriber processes of the failure and having them hold all transmissions. For other faults, or after a particular software fault occurs a predetermined number of times, the SRM may then require the name server / BOM to

notify all subscriber processes of the termination of the failed process. Alternatively, if a terminated process does not re-register within a predetermined amount of time, the name server / BOM may then notify all subscriber processes of the termination of the failed process.

Configuration Change:

Over time the user will likely make hardware changes to the computer system that require configuration changes. For example, the user may plug a fiber or cable (i.e., network connection) into an as yet unused port, in which case, the port must be enabled and, if not already enabled, then the port's line card must also be enabled. As other examples, the user may add another path to an already enabled port that was not fully utilized, and the user may add another line card to the computer system. Many types of configuration changes are possible, and the modular software architecture allows them to be made while the computer system is running (hot changes). Configuration changes may be automatically copied to persistent storage as they are made so that if the computer system is shut down and rebooted, the memory and configuration database will reflect the last known state of the hardware.

To make a configuration change, the user informs the NMS (e.g., NMS client 850a, Fig. 2a) of the particular change, and similar to the process for initial configuration, the NMS (e.g., NMS server 851a, Fig. 2a) changes the appropriate tables in the configuration database (copied to the NMS database) to implement the change.

Referring to Fig. 17, in one example of a configuration change, the user notifies the NMS that an additional path will be carried by SONET fiber 70c connected to port 44c. A new service endpoint (SE) 164 and a new ATM interface 166 are needed to handle the new path. The NMS adds a new record (row 168, Fig. 14a) to service endpoint table (SET) 76 to include service endpoint 10 corresponding to port physical identification number (PID) 1502 (port 44c). The NMS also adds a new record (row 170, Fig. 14e) to ATM instance table 114 to include ATM interface (IF) 12 corresponding to ATM group 3 and SE 10. Configuration database 42 may automatically copy the changes made to SET 76 and ATM instance table 114 to persistent storage 21 such that if the computer system is shut down and rebooted, the changes to the configuration database will be maintained.

Configuration database 42 also notifies (through the active query process) SEM 96c that a new service endpoint (SE 10) was added to the SET corresponding to its port (PID 1502), and configuration database 42 also notifies ATM instantiation 112 that a new ATM interface (ATM-IF 166) was added to the ATM interface table corresponding to ATM group 3. ATM 112 establishes ATM interface 166 and SEM 96c notifies port driver 142 that it has been assigned SE10. A communication link is established through NS 220b. Device driver 142 generates a service endpoint name using the assigned SE number and publishes this name and its process identification number with NS 220b. ATM interface 166 generates the same service endpoint name and subscribes to NS 220b for that service endpoint name. NS 220b provides ATM interface 166 with the process identification assigned to DD 142 allowing ATM interface 166 to communicate with device driver 142.

Certain board changes to computer system 10 are also configuration changes. After power-up and configuration, a user may plug another board into an empty computer system slot or remove an enabled board and replace it with a different board. In the case where applications and drivers for a line card added to computer system 10 are already loaded, the configuration change is similar to initial configuration. The additional line card may be identical to an already enabled line card, for example, line card 16a or if the additional line card requires different drivers (for different components) or different applications (e.g., IP), the different drivers and applications are already loaded because computer system 10 expects such cards to be inserted.

Referring to Fig. 18, while computer system 10 is running, when another line card 168 is inserted, master MCD 38 detects the insertion and communicates with a diagnostic program 170 being executed by the line card's processor 172 to learn the card's type and version number. MCD 38 uses the information it retrieves to update card table 47 and port table 49. MCD 38 then searches physical module description (PMD) file 48 in memory 40 for a record that matches the retrieved card type and version number and retrieves the name of the mission kernel image executable file (MKI.exe) that needs to be loaded on line card 168. Once determined, master MCD 38 passes the name of the MKI executable file to master SRM 36. SRM 36 downloads MKI executable file 174 from persistent storage 21 and passes it to a slave

SRM 176 running on line card 168. The slave SRM executes the received MKI executable file.

Referring to Fig. 19, slave MCD 178 then searches PMD file 48 in memory 40 on central processor 12 for a match with its line card's type and version number to find the names of all the device driver executable files associated needed by its line card. Slave MCD 178 provides these names to slave SRM 176 which then downloads and executes the device driver executable files (DD.exe) 180 from memory 40.

When master MCD 38 updates card table 47, configuration database 42 updated NMS database 61 which sends NMS 60 (e.g., NMS Server 851a, Fig. 2a) a notification of the change including card type and version number, the slot number into which the card was inserted and the physical identification (PID) assigned to the card by the master MCD. The NMS is updated, assigns an LID and updates the logical to physical table and notifies the user of the new hardware. The user then tells the NMS how to configure the new hardware, and the NMS implements the configuration change as described above for initial configuration.

Logical Model Change:

Where applications and device drivers for a new line card are not already loaded and where changes or upgrades to already loaded applications and device drivers are needed, logical model 280 (Figs. 3a-3b) must be changed and new view ids and APIs, NMS JAVA interface files, persistent layer metadata files and new DDL files must be re-generated. Software model 286 is changed to include models of the new or upgraded software, and hardware model 284 is changed to include models of any new hardware. New logical model 280' is then used by code generation system 336 to re-generate view ids and APIs for each application, including any new applications, for example, ATM version two 360, or device drivers, for example, device driver 362, and to re-generate DDL files 344' and 348' including new SQL commands and data relevant to the new hardware and/or software. The new logical model is also used to generate new NMS JAVA interface files 347' and new persistent layer metadata files 349'. Each application, including any new applications or drivers, is then pulled into the build process and links in a corresponding view id and API. The new applications and/or device drivers, NMS JAVA interface files, new persistent layer metadata files

and the new DDL files as well as any new hardware are then sent to the user of computer system 10.

New and upgraded applications and device drivers are being used by way of an example, and it should be understood that other processes, for example, modular system services and new Mission Kernel Images (MKIs), may be changed or upgraded in the same fashion.

Referring to Fig. 20, the user instructs the NMS to download the new applications and/or device drivers, for example, ATM version two 360 and device driver 362, as well as the new DDL files, for example, DDL files 344' and 348', into memory on work station 62. The NMS uses new NMS database DDL file 348' to upgrade NMS database 61 into new NMS database 61'. Alternatively, a new NMS database may be created using DDL file 348' and both databases temporarily maintained.

Application Upgrade:

For new applications and application upgrades, the NMS works with a software management system (SMS) service to implement the change while the computer system is running (hot upgrades or additions). The SMS is one of the modular system services, and like the MCD and the SRM, the SMS is a distributed application. Referring to Fig. 20, a master SMS 184 is executed by central processor 12 while slave SMSs 186a-186n are executed on each board.

Upgrading a distributed application that is running on multiple boards is more complicated than upgrading an application running on only one board. As an example of a distributed application upgrade, the user may want to upgrade all ATM applications running on various boards in the system using new ATM version two 360. This is by way of example, and it should be understood, that only one ATM application may be upgraded so long as it is compatible with the other versions of ATM running on other boards. ATM version two 360 may include many sub-processes, for example, an upgraded ATM application executable file (ATMv2.exe 189), an upgraded ATM control executable file (ATMv2_cntrl.exe 190) and an ATM configuration control file (ATMv2_cnfg_cntrl.exe). The NMS downloads

ATMv2.exe 189, ATMv2_cntrl.exe and ATMv2_cnfg_cntrl.exe to memory 40 on central processor 12.

The NMS then writes a new record into SMS table 192 indicating the scope of the configuration update. The scope of an upgrade may be indicated in a variety of ways. In one embodiment, the SMS table includes a field for the name of the application to be changed and other fields indicating the changes to be made. In another embodiment, the SMS table includes a revision number field 194 (Fig. 21) through which the NMS can indicate the scope of the change. Referring to Fig. 21, the right most position in the revision number may indicate, for example, the simplest configuration update (e.g., a bug fix), in this case, termed a "service update level" 196. Any software revisions that differ by only the service update level can be directly applied without making changes in the configuration database or API changes between the new and current revision. The next position may indicate a slightly more complex update, in this case, termed a "subsystem compatibility level" 198. These changes include changes to the configuration database and/or an API. The next position may indicate a "minor revision level" 200 update indicating more comprehensive changes in both the configuration database and one or more APIs. The last position may indicate a "major revision level" 202 update indicative of wholesale changes in multiple areas and may require a reboot of the computer system to implement. For a major revision level change, the NMS will download a complete image including a kernel image.

During initial configuration, the SMS establishes an active query on SMS table 192. Consequently, when the NMS changes the SMS table, the configuration database sends a notification to master SMS 184 including the change. In some instances, the change to an application may require changes to configuration database 42. The SMS determines the need for configuration conversion based on the scope of the release or update. If the configuration database needs to be changed, then the software, for example, ATM version two 360, provided by the user and downloaded by the NMS also includes a configuration control executable file, for example, ATMv2_cnfig_cntrl.exe 191, and the name of this file will be in the SMS table record. The master SMS then directs slave SRM 37a on central processor 12 to execute the configuration control file which uses DDL file 344' to upgrade old

configuration database 42 into new configuration database 42' by creating new tables, for example, ATM group table 108' and ATM interface table 114'.

Existing processes using their view ids and APIs to access new configuration database 42' in the same manner as they accessed old configuration database 42. However, when new processes (e.g., ATM version two 360 and device driver 362) access new configuration database 42', their view ids and APIs allow them to access new tables and data within new configuration database 42'.

The master SMS also reads ATM group table 108' to determine that instances of ATM are being executed on line cards 16a-16n. In order to upgrade a distributed application, in this instance, ATM, the Master SMS will use a lock step procedure. Master SMS 184 tells each slave SMS 186b-186n to stall the current versions of ATM. When each slave responds, Master SMS 184 then tells slave SMSs 186b-186n to download and execute ATMv2_cntrl.exe 190 from memory 40. Upon instructions from the slave SMSs, slave SRMs 37b-37n download and execute copies of ATMv2_cntrl.exe 204a-204n. The slave SMSs also pass data to the ATMv2cntrl.exe file through the SRM. The data instructs the control shim to start in upgrade mode and passes required configuration information. The upgraded ATMv2 controllers 204a-204n then use ATM group table 108' and ATM interface table 114' as described above to implement ATMv2 206a-206n on each of the line cards. In this example, each ATM controller is shown implementing one instance of ATM on each line card, but as explained below, the ATM controller may implement multiple instances of ATM on each line card.

As part of the upgrade mode, the updated versions of ATMv2 206a-206n retrieve active state from the current versions of ATM 188a-188n. The retrieval of active state can be accomplished in the same manner that a redundant or backup instantiation of ATM retrieves active state from the primary instantiation of ATM. When the upgraded instances of ATMv2 are executing and updated with active state, the ATMv2 controllers notify the slave SMSs 186b-186n on their board and each slave SMS 186b-186n notifies master SMS 184. When all boards have notified the master SMS, the master SMS tells the slave SMSs to switchover to ATMv2 206a-206n. The slave SMSs tell the slave SRMs running on their board, and the slave SRMs transition

the new ATMv2 processes to the primary role. This is termed "lock step upgrade" because each of the line cards is switched over to the new ATMv2 processes simultaneously.

There may be upgrades that require changes to multiple applications and to the APIs for those applications. For example, a new feature may be added to ATM that also requires additional functionality to be added to the Multi-Protocol Label Switching (MPLS) application. The additionally functionality may change the peer-to-peer API for ATM, the peer-to-peer API for MPLS and the API between ATM and MPLS. In this scenario, the upgrade operation must avoid allowing the "new" version of ATM to communicate with itself or the "old" version of MPLS and vice versa. The master SMS will use the release number scheme to determine the requirements for the individual upgrade. For example, the upgrade may be from release 1.0.0.0 to 1.0.1.3 where the release differs by the subsystem compatibility level. The SMS implements the upgrade in a lock step fashion. All instances of ATM and MPLS are upgraded first. The slave SMS on each line card then directs the slave SRM on its board to terminate all "old" instances of ATM and MPLS and switchover to the new instances of MPLS and ATM. The simultaneous switchover to new versions of both MPLS and ATM eliminate any API compatibility errors.

Referring to Fig. 22, instead of directly upgrading configuration database 42 on central processor 12, a backup configuration database 420 on a backup central processor 13 may be upgraded first. As described above, computer system 10 includes central processor 12. Computer system 10 may also include a redundant or backup central processor 13 that mirrors or replicates the active state of central processor 12. Backup central processor 13 is generally in stand-by mode unless central processor 12 fails at which point a fail-over to backup central processor 13 is initiated to allow the backup central processor to be substituted for central processor 12. In addition to failures, backup central processor 13 may be used for software and hardware upgrades that require changes to the configuration database. Through backup central processor 13, upgrades can be made to backup configuration database 420 instead of to configuration database 42.

The upgrade is begun as discussed above with the NMS downloading ATM version two 360 – including ATMv2.exe 189, ATMv2_cntrl.exe and ATMv2_cnfg_cntrl.exe – and DDL file 344' to memory on central processor 12. Simultaneously, because central processor 13 is in backup mode, the application and DDL file are also copied to memory on central processor 13. The NMS also creates a software load record in SMS table 192, 192' indicating the upgrade. In this embodiment, when the SMS determines that the scope of the upgrade requires an upgrade to the configuration database, the master SMS instructs slave SMS 186e on central processor 13 to perform the upgrade. Slave SMS 186e works with slave SRM 37e to cause backup processor 13 to change from backup mode to upgrade mode.

In upgrade mode, backup processor 13 stops replicating the active state of central processor 12. Any changes made to new configuration database 420 are copied to new NMS database 61'. Slave SMS 186e then directs slave SRM 37e to execute the configuration control file which uses DDL file 344' to upgrade configuration database 420.

Once configuration database 420 is upgraded, a fail-over or switch-over from central processor 12 to backup central processor 13 is initiated. Central processor 13 then begins acting as the primary central processor and applications running on central processor 13 and other boards throughout computer system 10 begin using upgraded configuration database 420.

Central processor 12 may not become the backup central processor right away. Instead, central processor 12 with its older copy of configuration database 42 stays dormant in case an automatic downgrade is necessary (described below). If the upgrade goes smoothly and is committed (described below), then central processor 12 will begin operating in backup mode and replace old configuration database 42 with new configuration database 420.

Device Driver Upgrade:

Device driver software may also be upgraded and the implementation of device driver upgrades is similar to the implementation of application upgrades. The user informs the NMS of the device driver change and provides a copy of the new software (e.g.,

DD^.exe 362, Figs. 20 and 23). The NMS downloads the new device driver to memory 40 on central processor 12, and the NMS writes a new record in SMS table 192 indicating the device driver upgrade. Configuration database 42 sends a notification to master SMS 184 including the name of the driver to be upgraded. To determine where the original device driver is currently running in computer system 10, the master SMS searches PMD file 48 for a match of the device driver name (existing device driver, not upgraded device driver) to learn with which module type and version number the device driver is associated. The device driver may be running on one or more boards in computer system 10. As described above, the PMD file corresponds the module type and version number of a board with the mission kernel image for that board as well as the device drivers for that board. The SMS then searches card table 47 for a match with the module type and version number found in the PMD file. Card table 47 includes records corresponding module type and version number with the physical identification (PID) and slot number of that board. The master SMS now knows the board or boards within computer system 10 on which to load the upgraded device driver. If the device driver is for a particular port, then the SMS must also search the port table to learn the PID for that port.

The master SMS notifies each slave SMS running on boards to be upgraded of the name of the device driver executable file to download and execute. In the example, master SMS 184 sends slave SMS 186f the name of the upgraded device driver (DD^.exe 362) to download. Slave SMS 186f tells slave SRM to download and execute DD^.exe 362 in upgrade mode. Once downloaded, DD^.exe 363 (copy of DD^.exe 362) gathers active state information from the currently running DD.exe 212 in a similar fashion as a redundant or backup device driver would gather active state. DD^.exe 362 then notifies slave SRM 37f that active state has been gathered, and slave SRM 37f stops the current DD.exe 212 process and transitions the upgraded DD^.exe 362 process to the primary role.

Automatic Downgrade:

Often, implementation of an upgrade, can cause unexpected errors in the upgraded software, in other applications or in hardware. As described above, a new configuration database 42' (Fig. 20) is generated and changes to the new configuration database are made in new tables (e.g., ATM interface table 114' and

ATM group table 108', Fig. 20) and new executable files (e.g., ATMv2.exe 189, ATMv2_cntrl.exe 190 and ATMv2_cnfg_cntrl.exe 191) are downloaded to memory 40. Importantly, the old configuration database records and the original application files are not deleted or altered. In the embodiment where changes are made directly to configuration database 42 on central processor 12, they are made only in non-persistent memory until committed (described below). In the embodiment where changes are made to backup configuration database 420 on backup central processor 13, original configuration database 42 remains unchanged.

Because the operating system provides a protected memory model that assigns different process blocks to different processes, including upgraded applications, the original applications will not share memory space with the upgraded applications and, therefore, cannot corrupt or change the memory used by the original application. Similarly, memory 40 is capable of simultaneously maintaining the original and upgraded versions of the configuration database records and executable files as well as the original and upgraded versions of the applications (e.g., ATM 188a-188n). As a result, the SMS is capable of an automatic downgrade on the detection of an error. To allow for automatic downgrade, the SRMs pass error information to the SMS. The SMS may cause the system to revert to the old configuration and application (i.e., automatic downgrade) on any error or only for particular errors.

As mentioned, often upgrades to one application may cause unexpected faults or errors in other software. If the problem causes a system shut down and the configuration upgrade was stored in persistent storage, then the system, when powered back up, will experience the error again and shut down again. Since, the upgrade changes to the configuration database are not copied to persistent storage 21 until the upgrade is committed, if the computer system is shut down, when it is powered back up, it will use the original version of the configuration database and the original executable files, that is, the computer system will experience an automatic downgrade.

Additionally, a fault induced by an upgrade may cause the system to hang, that is, the computer system will not shut down but will also become inaccessible by the NMS and inoperable. To address this concern, in one embodiment, the NMS and the master

SMS periodically send messages to each other indicating they are executing appropriately. If the SMS does not receive one of these messages in a predetermined period of time, then the SMS knows the system has hung. The master SMS may then tell the slave SMSs to revert to the old configuration (i.e., previously executing copies of ATM 188a-188n) and if that does not work, the master SMS may re-start / re-boot computer system 10. Again, because the configuration changes were not saved in persistent storage, when the computer system powers back up, the old configuration will be the one implemented.

Evaluation Mode:

Instead of implementing a change to a distributed application across the entire computer system, an evaluation mode allows the SMS to implement the change in only a portion of the computer system. If the evaluation mode is successful, then the SMS may fully implement the change system wide. If the evaluation mode is unsuccessful, then service interruption is limited to only that portion of the computer system on which the upgrade was deployed. In the above example, instead of executing the upgraded ATMv2 189 on each of the line cards, the ATMv2 configuration convert file 191 will create an ATMv2 group table 108' indicating an upgrade only to one line card, for example, line card 16a. Moreover, if multiple instantiations of ATM are running on line card 16a (e.g., one instantiation per port), the ATMv2 configuration convert file may indicate through ATMv2 interface table 114' that the upgrade is for only one instantiation (e.g., one port) on line card 16a. Consequently, a failure is likely to only disrupt service on that one port, and again, the SMS can further minimize the disruption by automatically downgrading the configuration of that port on the detection of an error. If no error is detected during the evaluation mode, then the upgrade can be implemented over the entire computer system.

Upgrade Commitment:

Upgrades are made permanent by saving the new application software and new configuration database and DDL file in persistent storage and removing the old configuration data from memory 40 as well as persistent storage. As mentioned above, changes may be automatically saved in persistent storage as they are made in non-persistent memory (no automatic downgrade), or the user may choose to

automatically commit an upgrade after a successful time interval lapses (evaluation mode). The time interval from upgrade to commitment may be significant. During this time, configuration changes may be made to the system. Since these changes are typically made in non-persistent memory, they will be lost if the system is rebooted prior to upgrade commitment. Instead, to maintain the changes, the user may request that certain configuration changes made prior to upgrade commitment be copied into the old configuration database in persistent memory. Alternatively, the user may choose to manually commit the upgrade at his or her leisure. In the manual mode, the user would ask the NMS to commit the upgrade and the NMS would inform the master SMS, for example, through a record in the SMS table.

Independent Process Failure and Restart:

Depending upon the fault policy managed by the slave SRMs on each board, the failure of an application or device driver may not immediately cause an automatic downgrade during an upgrade process. Similarly, the failure of an application or device driver during normal operation may not immediately cause the fail over to a backup or redundant board. Instead, the slave SRM running on the board may simply restart the failing process. After multiple failures by the same process, the fault policy may cause the SRM to take more aggressive measures such as automatic downgrade or fail-over.

Referring to Fig. 24, if an application, for example, ATM application 230 fails, the slave SRM on the same board as ATM 230 may simply restart it without having to reboot the entire system. As described above, under the protected memory model, a failing process cannot corrupt the memory blocks used by other processes. Typically, an application and its corresponding device drivers would be part of the same memory block or even part of the same software program, such that if the application failed, both the application and device drivers would need to be restarted. Under the modular software architecture, however, applications, for example ATM application 230, are independent of the device drivers, for example, ATM driver 232 and Device Drivers (DD) 234a-234c. This separation of the data plane (device drivers) and control plane (applications) results in the device drivers being peers of the applications. Hence, while the ATM application is terminated and restarted, the device drivers continue to function.

For network devices, this separation of the control plane and data plane means that the connections previously established by the ATM application are not lost when ATM fails and hardware controlled by the device drivers continue to pass data through connections previously established by the ATM application. Until the ATM application is restarted and re-synchronized (e.g., through an audit process, described below) with the active state of the device drivers, no new network connections may be established but the device drivers continue to pass data through the previously established connections to allow the network device to minimize disruption and maintain high availability.

Local Backup:

If a device driver, for example, device driver 234, fails instead of an application, for example, ATM 230, then data cannot be passed. For a network device, it is critical to continue to pass data and not lose network connections. Hence, the failed device driver must be brought back up (i.e., recovered) as soon as possible. In addition, the failing device driver may have corrupted the hardware it controls, therefore, that hardware must be reset and reinitialized. The hardware may be reset as soon as the device driver terminates or the hardware may be reset later when the device driver is restarted.

Resetting the hardware stops data flow. In some instances, therefore, resetting the hardware will be delayed until the device driver is restarted to minimize the time period during which data is not flowing. Alternatively, the failing device driver may have corrupted the hardware, thus, resetting the hardware as soon as the device driver is terminated may be important to prevent data corruption. In either case, the device driver re-initializes the hardware during its recovery.

Again, because applications and device drivers are assigned independent memory blocks, a failed device driver can be restarted without having to restart associated applications and device drivers. Independent recovery may save significant time as described above for applications. In addition, restoring the data plane (i.e., device drivers) can be simpler and faster than restoring the control plane (i.e., applications). While it may be just as challenging in terms of raw data size, device driver recovery may simply require that critical state data be copied into place in a few large blocks,

as opposed to application recovery which requires the successive application of individual configuration elements and considerable parsing, checking and analyzing. In addition, the application may require data stored in the configuration database on the central processor or data stored in the memory of other boards. The configuration database may be slow to access especially since many other applications also access this database. The application may also need time to access a management information base (MIB) interface.

To increase the speed with which a device driver is brought back up, the restarted device driver program accesses local backup 236. In one example, local backup is a simple storage/retrieval process that maintains the data in simple lists in physical memory (e.g., random access memory, RAM) for quick access. Alternatively, local backup may be a database process, for example, a Polyhedra database, similar to the configuration database.

Local backup 236 stores the last snap shot of critical state information used by the original device driver before it failed. The data in local backup 236 is in the format required by the device driver. In the case of a network device, local back up data may include path information, for example, service endpoint, path width and path location. Local back up data may also include virtual interface information, for example, which virtual interfaces were configured on which paths and virtual circuit (VC) information, for example, whether each VC is switched or passed through segmentation and reassembly (SAR), whether each VC is a virtual channel or virtual path and whether each VC is multicast or merge. The data may also include traffic parameters for each VC, for example, service class, bandwidth and/or delay requirements.

Using the data in the local backup allows the device driver to quickly recover. An Audit process resynchronizes the restarted device driver with associated applications and other device drivers such that the data plane can again transfer network data. Having the backup be local reduces recovery time. Alternatively, the backup could be stored remotely on another board but the recovery time would be increased by the amount of time required to download the information from the remote location.

Audit Process:

It is virtually impossible to ensure that a failed process is synchronized with other processes when it restarts, even when backup data is available. For example, an ATM application may have set up or torn down a connection with a device driver but the device driver failed before it updated corresponding backup data. When the device driver is restarted, it will have a different list of established connections than the corresponding ATM application (i.e., out of synchronization). The audit process allows processes like device drivers and ATM applications to compare information, for example, connection tables, and resolve differences. For instance, connections included in the driver's connection table and not in the ATM connection table were likely torn down by ATM prior to the device driver crash and are, therefore, deleted from the device driver connection table. Connections that exist in the ATM connection table and not in the device driver connection table were likely set up prior to the device driver failure and may be copied into the device driver connection table or deleted from the ATM connection table and re-set up later. If an ATM application fails and is restarted, it must execute an audit procedure with its corresponding device driver or drivers as well as with other ATM applications since this is a distributed application.

Vertical Fault Isolation:

Typically, a single instance of an application executes on a single card or in a system. Fault isolation, therefore, occurs at the card level or the system level, and if a fault occurs, an entire card – and all the ports on that card – or the entire system – and all the ports in the system -- is affected. In a large communications platform, thousands of customers may experience service outages due to a single process failure.

For resiliency and fault isolation one or more instances of an application and/or device driver may be started per port on each line card. Multiple instances of applications and device drivers are more difficult to manage and require more processor cycles than a single instance of each but if an application or device driver fails, only the port those processes are associated with is affected. Other applications and associated ports – as well as the customers serviced by those ports -- will not experience service outages. Similarly, a hardware failure associated with only one

port will only affect the processes associated with that port. This is referred to as vertical fault isolation.

Referring to Fig. 25, as one example, line card 16a is shown to include four vertical stacks 400, 402, 404, and 406. Vertical stack 400 includes one instance of ATM 110 and one device driver 43a and is associated with port 44a. Similarly, vertical stacks 402, 404 and 406 include one instance of ATM 111, 112, 113 and one device driver 43b, 43c, 43d, respectively and each vertical stack is associated with a separate port 44b, 44c, 44d, respectively. If ATM 112 fails, then only vertical stack 404 and its associated port 44c are affected. Service is not disrupted on the other ports (ports 44a, 44b, 44d) since vertical stacks 400, 402, and 406 are unaffected and the applications and drivers within those stacks continue to execute and transmit data. Similarly, if device driver 43b fails, then only vertical stack 402 and its associated port 44b are affected.

Vertical fault isolation allows processes to be deployed in a fashion supportive of the underlying hardware architecture and allows processes associated with particular hardware (e.g., a port) to be isolated from processes associated with other hardware (e.g., other ports) on the same or a different line card. Any single hardware or software failure will affect only those customers serviced by the same vertical stack. Vertical fault isolation provides a fine grain of fault isolation and containment. In addition, recovery time is reduced to only the time required to re-start a particular application or driver instead of the time required to re-start all the processes associated with a line card or the entire system.

Fault / Event Detection:

Traditionally, fault detection and monitoring does not receive a great deal of attention from network equipment designers. Hardware components are subjected to a suite of diagnostic tests when the system powers up. After that, the only way to detect a hardware failure is to watch for a red light on a board or wait for a software component to fail when it attempts to use the faulty hardware. Software monitoring is also reactive. When a program fails, the operating system usually detects the failure and records minimal debug information.

Current methods provide only sporadic coverage for a narrow set of hard faults. Many subtler failures and events often go undetected. For example, hardware components sometimes suffer a minor deterioration in functionality, and changing network conditions stress the software in ways that were never expected by the designers. At times, the software may be equipped with the appropriate instrumentation to detect these problems before they become hard failures, but even then, network operators are responsible for manually detecting and repairing the conditions.

Systems with high availability goals must adopt a more proactive approach to fault and event monitoring. In order to provide comprehensive fault and event detection, different hierarchical levels of fault/event management software are provided that intelligently monitor hardware and software and proactively take action in accordance with a defined fault policy. A fault policy based on hierarchical scopes ensures that for each particular type of failure the most appropriate action is taken. This is important because over-reacting to a failure, for example, re-booting an entire computer system or re-starting an entire line card, may severely and unnecessarily impact service to customers not affected by the failure, and under-reacting to failures, for example, restarting only one process, may not completely resolve the fault and lead to additional, larger failures. Monitoring and proactively responding to events may also allow the computer system and network operators to address issues before they become failures. For example, additional memory may be assigned to programs or added to the computer system before a lack of memory causes a failure.

Hierarchical Scopes and Escalation:

Referring to Fig. 26, in one embodiment, master SRM 36 serves as the top hierarchical level fault/event manager, each slave SRM 37a-37n serves as the next hierarchical level fault/event manager, and software applications resident on each board, for example, ATM 110-113 and device drivers 43a-43d on line card 16a include sub-processes that serve as the lowest hierarchical level fault/event managers (i.e., local resiliency managers, LRM). Master SRM 36 downloads default fault policy (DFP) files (metadata) 430a-430n from persistent storage to memory 40. Master SRM 36 reads a master default fault policy file (e.g., DFP 430a) to understand its fault policy, and each slave SRM 37a-37n downloads a default fault policy file

(e.g., DFP 430b-430n) corresponding to the board on which the slave SRM is running. Each slave SRM then passes to each LRM a fault policy specific to each local process.

A master logging entity 431 also runs on central processor 12 and slave logging entities 433a-433n run on each board. Notifications of failures and other events are sent by the master SRM, slave SRMs and LRMs to their local logging entity which then notifies the master logging entity. The master logging entity enters the event in a master event log file 435. Each local logging entity may also log local events in a local event log file 435a-435n.

In addition, a fault policy table 429 may be created in configuration database 42 by the NMS when the user wishes to over-ride some or all of the default fault policy (see configurable fault policy below), and the master and slave SRMs are notified of the fault policies through the active query process.

Referring to Fig. 27, as one example, ATM application 110 includes many sub-processes including, for example, an LRM program 436, a Private Network-to-Network Interface (PNNI) program 437, an Interim Link Management Interface (ILMI) program 438, a Service Specific Connection Oriented Protocol (SSCOP) program 439, and an ATM signaling (SIG) program 440. ATM application 110 may include many other sub-programs only a few have been shown for convenience. Each sub-process may also include sub-processes, for example, ILMI sub-processes 438a-438n. In general, the upper level application (e.g., ATM 110) is assigned a process memory block that is shared by all its sub-processes.

If, for example, SSCOP 439 detects a fault, it notifies LRM 436. LRM 436 passes the fault to local slave SRM 37b, which catalogs the fault in the ATM application's fault history and sends a notice to local slave logging entity 433b. The slave logging entity sends a notice to master logging entity 431, which may log the event in master log event file 435. The local logging entity may also log the failure in local event log 435a. LRM 436 also determines, based on the type of failure, whether it can fully resolve the error and do so without affecting other processes outside its scope, for example, ATM 111-113, device drivers 43a-43d and their sub-processes and

processes running on other boards. If yes, then the LRM takes corrective action in accordance with its fault policy. Corrective action may include restarting SSCOP 439 or resetting it to a known state.

Since all sub-processes within an application, including the LRM sub-process, share the same memory space, it may be insufficient to restart or reset a failing sub-process (e.g., SSCOP 439). Hence, for most failures, the fault policy will cause the LRM to escalate the failure to the local slave SRM. In addition, many failures will not be presented to the LRM but will, instead, be presented directly to the local slave SRM. These failures are likely to have been detected by either processor exceptions, OS errors or low-level system service errors. Instead of failures, however, the sub-processes may notify the LRM of events that may require action. For example, the LRM may be notified that the PNNI message queue is growing quickly. The LRM's fault policy may direct it to request more memory from the operating system. The LRM will also pass the event to the local slave SRM as a non-fatal fault. The local slave SRM will catalog the event and log it with the local logging entity, which may also log it with the master logging entity. The local slave SRM may take more severe action to recover from an excessive number of these non-fatal faults that result in memory requests.

If the event or fault (or the actions required to handle either) will affect processes outside the LRM's scope, then the LRM notifies slave SRM 37b of the event or failure. In addition, if the LRM detects and logs the same failure or event multiple times and in excess of a predetermined threshold set within the fault policy, the LRM may escalate the failure or event to the next hierarchical scope by notifying slave SRM 37b. Alternatively or in addition, the slave SRM may use the fault history for the application instance to determine when a threshold is exceeded and automatically execute its fault policy.

When slave SRM 37b detects or is notified of a failure or event, it notifies slave logging entity 435b. The slave logging entity notifies master logging entity 431, which may log the failure or event in master event log 435, and the slave logging entity may also log the failure or event in local event log 435b. Slave SRM 37b also determines, based on the type of failure or event, whether it can handle the error

without affecting other processes outside its scope, for example, processes running on other boards. If yes, then slave SRM 37b takes corrective action in accordance with its fault policy and logs the fault. Corrective action may include re-starting one or more applications on line card 16a.

If the fault or recovery actions will affect processes outside the slave SRM's scope, then the slave SRM notifies master SRM 36. In addition, if the slave SRM has detected and logged the same failure multiple times and in excess of a predetermined threshold, then the slave SRM may escalate the failure to the next hierarchical scope by notifying master SRM 36 of the failure. Alternatively, the master SRM may use its fault history for a particular line card to determine when a threshold is exceeded and automatically execute its fault policy.

When master SRM 36 detects or receives notice of a failure or event, it notifies slave logging entity 433a, which notifies master logging entity 431. The master logging entity 431 may log the failure or event in master log file 435 and the slave logging entity may log the failure or event in local event log 435a. Master SRM 36 also determines the appropriate corrective action based on the type of failure or event and its fault policy. Corrective action may require failing-over one or more line cards 16a-16n or other boards, including central processor 12, to redundant backup boards or, where backup boards are not available, simply shutting particular boards down. Some failures may require the master SRM to re-boot the entire computer system.

An example of a common error is a memory access error. As described above, when the slave SRM starts a newinstance of an application, it requests a protected memory block from the local operating system. The local operating systems assign each instance of an application one block of local memory and then program the local memory management unit (MMU) hardware with which processes have access (read and/or write) to each block of memory. An MMU detects a memory access error when a process attempts to access a memory block not assigned to that process. This type of error may result when the process generates an invalid memory pointer. The MMU prevents the failing process from corrupting memory blocks used by other processes (i.e., protected memory model) and sends a hardware exception to the local processor. A local operating system fault handler detects the hardware exception and

determines which process attempted the invalid memory access. The fault handler then notifies the local slave SRM of the hardware exception and the process that caused it. The slave SRM determines the application instance within which the fault occurred and then goes through the process described above to determine whether to take corrective action, such as restarting the application, or escalate the fault to the master SRM.

As another example, a device driver, for example, device driver 43a may determine that the hardware associated with its port, for example, port 44a, is in a bad state. Since the failure may require the hardware to be swapped out or failed-over to redundant hardware or the device driver itself to be re-started, the device driver notifies slave SRM 37b. The slave SRM then goes through the process described above to determine whether to take corrective action or escalate the fault to the master SRM.

As a third example, if a particular application instance repeatedly experiences the same software error but other similar application instances running on different ports do not experience the same error, the slave SRM may determine that it is likely a hardware error. The slave SRM would then notify the master SRM which may initiate a fail-over to a backup board or, if no backup board exists, simply shut down that board or only the failing port on that board. Similarly, if the master SRM receives failure reports from multiple boards indicating Ethernet failures, the master SRM may determine that the Ethernet hardware is the problem and initiate a fail-over to backup Ethernet hardware.

Consequently, the failure type and the failure policy determine at what scope recovery action will be taken. The higher the scope of the recovery action, the larger the temporary loss of services. Speed of recovery is one of the primary considerations when establishing a fault policy. Restarting a single software process is much faster than switching over an entire board to a redundant board or re-booting the entire computer system. When a single process is restarted, only a fraction of a card's services are affected. Allowing failures to be handled at appropriate hierarchical levels avoids unnecessary recovery actions while ensuring that sufficient recovery actions are taken, both of which minimize service disruption to customers.

Hierarchical Descriptors:

Hierarchical descriptors may be used to provide information specific to each failure or event. The hierarchical descriptors provide granularity with which to report faults, take action based on fault history and apply fault recovery policies. The descriptors can be stored in master event log file 435 or local event log files 435a-435n through which faults and events may be tracked and displayed to the user and allow for fault detection at a fine granular level and proactive response to events. In addition, the descriptors can be matched with descriptors in the fault policy to determine the recovery action to be taken.

Referring to Fig. 28, in one embodiment, a descriptor 441 includes a top hierarchical class field 442, a next hierarchical level sub-class field 444, a lower hierarchical level type field 446 and a lowest level instance field 448. The class field indicates whether the failure or event is related (or suspected to relate) to hardware or software. The subclass field categorizes events and failures into particular hardware or software groups. For example, under the hardware class, subclass indications may include whether the fault or event is related to memory, Ethernet, switch fabric or network data transfer hardware. Under the software class, subclass indications may include whether the fault or event is a system fault, an exception or related to a specific application, for example, ATM.

The type field more specifically defines the subclass failure or event. For example, if a hardware class, Ethernet subclass failure has occurred, the type field may indicate a more specific type of Ethernet failure, for instance, a cyclic redundancy check (CRC) error or a runt packet error. Similarly, if a software class, ATM failure or event has occurred, the type field may indicate a more specific type of ATM failure or event, for instance, a private network-to-network interface (PNNI) error or a growing message queue event. The instance field identifies the actual hardware or software that failed or generated the event. For example, with regard to a hardware class, Ethernet subclass, CRC type failure, the instance indicates the actual Ethernet port that experienced the failure. Similarly, with regard to a software class, ATM subclass, PNNI type, the instance indicates the actual PNNI sub-program that experienced the failure or generated the event.

When a fault or event occurs, the hierarchical scope that first detects the failure or event creates a descriptor by filling in the fields described above. In some cases, however, the Instance field is not applicable. The descriptor is sent to the local logging entity, which may log it in the local event log file before notifying the master logging entity, which may log it in the master event log file 435. The descriptor may also be sent to the local slave SRM, which tracks fault history based on the descriptor contents per application instance. If the fault or event is escalated, then the descriptor is passed to the next higher hierarchical scope.

When slave SRM 37b receives the fault / event notification and the descriptor, it compares it to descriptors in the fault policy for the particular scope in which the fault occurred looking for a match or a best case match which will indicate the recovery procedure to follow. Fault descriptors within the fault policy can either be complete descriptors or have wildcards in one or more fields. Since the descriptors are hierarchical from left to right, wildcards in descriptor fields only make sense from right to left. The fewer the fields with wildcards, the more specific the descriptor. For example, a particular fault policy may apply to all software faults and would, therefore, include a fault descriptor having the class field set to "software" and the remaining fields - subclass, type, and instance - set to wildcard or "match all." The slave SRM searches the fault policy for the best match (i.e., the most fields matched) with the descriptor to determine the recovery action to be taken.

Configurable Fault Policy:

In actual use, a computer system is likely to encounter scenarios that differ from those in which the system was designed and tested. Consequently, it is nearly impossible to determine all the ways in which a computer system might fail, and in the face of an unexpected error, the default fault policy that was shipped with the computer system may cause the hierarchical scope (master SRM, slave SRM or LRM) to under-react or over-react. Even for expected errors, after a computer system ships, certain recovery actions in the default fault policy may be determined to be over aggressive or too lenient. Similar issues may arise as new software and hardware is released and/or upgraded.

A configurable fault policy allows the default fault policy to be modified to address behavior specific to a particular upgrade or release or to address behavior that was learned after the implementation was released. In addition, a configurable fault policy allows users to perform manual overrides to suit their specific requirements and to tailor their policies based on the individual failure scenarios that they are experiencing.

The modification may cause the hierarchical scope to react more or less aggressively to particular known faults or events, and the modification may add recovery actions to handle newly learned faults or events. The modification may also provide a temporary patch while a software or hardware upgrade is developed to fix a particular error.

If an application runs out of memory space, it notifies the operating system and asks for more memory. For certain applications, this is standard operating procedure. As an example, an ATM application may have set up a large number of virtual circuits and to continue setting up more, additional memory is needed. For other applications, a request for more memory indicates a memory leak error. The fault policy may require that the application be re-started causing some service disruption. It may be that re-starting the application eventually leads to the same error due to a bug in the software. In this instance, while a software upgrade to fix the bug is developed, a temporary patch to the fault policy may be necessary to allow the memory leak to continue and prevent repeated application re-starts that may escalate to line card re-start or fail-over and eventually to a re-boot of the entire computer system. A temporary patch to the default fault policy may simply allow the hierarchical scope, for example, the local resiliency manager or the slave SRM, to assign additional memory to the application. Of course, an eventual re-start of the application is likely to be required if the application's leak consumes too much memory.

A temporary patch may also be needed while a hardware upgrade or fix is developed for a particular hardware fault. For instance, under the default fault policy, when a particular hardware fault occurs, the recovery policy may be to fail-over to a backup board. If the backup board includes the same hardware with the same hardware bug, for example, a particular semiconductor chip, then the same error will occur on the backup board. To prevent a repetitive fail-over while a hardware fix is developed, the

temporary patch to the default fault policy may be to restart the device driver associated with the particular hardware instead of failing-over to the backup board.

In addition to the above needs, a configurable fault policy also allows purchasers of computer system 10 (e.g., network service providers) to define their own policies. For example, a network service provider may have a high priority customer on a particular port and may want all errors and events (even minor ones) to be reported to the NMS and displayed to the network manager. Watching all errors and events might give the network manager early notice of growing resource consumption and the need to plan to dedicate additional resources to this customer.

As another example, a user of computer system 10 may want to be notified when any process requests more memory. This may give the user early notice of the need to add more memory to their system or to move some customers to different line cards.

Referring again to Fig. 26, to change the default fault policy as defined by default fault policy (DFP) files 430a-430n, a configuration fault policy file 429 is created by the NMS in the configuration database. An active query notification is sent by the configuration database to the master SRM indicating the changes to the default fault policy. The master SRM notifies any slave SRMs of any changes to the default fault policies specific to the boards on which they are executing, and the slave SRMs notify any LRMs of any changes to the default fault policies specific to their process. Going forward, the default fault policies - as modified by the configuration fault policy - are used to detect, track and respond to events or failures.

Alternatively, active queries may be established with the configuration database for configuration fault policies specific to each board type such that the slave SRMs are notified directly of changes to their default fault policies.

A fault policy (whether default or configured) is specific to a particular scope and descriptor and indicates a particular recovery action to take. As one example, a temporary patch may be required to handle hardware faults specific to a known bug in an integrated circuit chip. The configured fault policy, therefore, may indicate a scope of all line cards, if the component is on all line cards, or only a specific type of

line card that includes that component. The configured fault policy may also indicate that it is to be applied to all hardware faults with that scope, for example, the class will indicate hardware (HW) and all other fields will include wildcards (e.g., HW.*.*.*). Instead, the configured fault policy may only indicate a particular type of hardware failure, for example, CRC errors on transmitted Ethernet packets (e.g., HW.Ethernet.TxCRC.*).

Redundancy:

As previously mentioned, a major concern for service providers is network downtime. In pursuit of "five 9's availability" or 99.999% network up time, service providers must minimize network outages due to equipment (i.e., hardware) and all too common software failures. Developers of computer systems often use redundancy measures to minimize downtime and enhance system resiliency. Redundant designs rely on alternate or backup resources to overcome hardware and/or software faults. Ideally, the redundancy architecture allows the computer system to continue operating in the face of a fault with minimal service disruption, for example, in a manner transparent to the service provider's customer.

Generally, redundancy designs come in two forms: 1:1 and 1:N. In a so-called "1:1 redundancy" design, a backup element exists for every active or primary element (i.e., hardware backup). In the event that a fault affects a primary element, a corresponding backup element is substituted for the primary element. If the backup element has not been in a "hot" state (i.e., software backup), then the backup element must be booted, configured to operate as a substitute for the failing element, and also provided with the "active state" of the failing element to allow the backup element to take over where the failed primary element left off. The time required to bring the software on the backup element to an "active state" is referred to as synchronization time. A long synchronization time can significantly disrupt system service, and in the case of a computer network device, if synchronization is not done quickly enough, then hundreds or thousands of network connections may be lost which directly impacts the service provider's availability statistics and angers network customers.

To minimize synchronization time, many 1:1 redundancy schemes support hot backup of software, which means that the software on the backup elements mirror the

software on the primary elements at some level. The "hotter" the backup element – that is, the closer the backup mirrors the primary – the faster a failed primary can be switched over or failed over to the backup. The "hottest" backup element is one that runs hardware and software simultaneously with a primary element conducting all operations in parallel with the primary element. This is referred to as a "1+1 redundancy" design and provides the fastest synchronization.

Significant costs are associated with 1:1 and 1+1 redundancy. For example, additional hardware costs may include duplicate memory components and printed circuit boards including all the components on those boards. The additional hardware may also require a larger supporting chassis. Space is often limited, especially in the case of network service providers who may maintain hundreds of network devices. Although 1:1 redundancy improves system reliability, it decreases service density and decreases the mean time between failures. Service density refers to the proportionality between the net output of a particular device and its gross hardware capability. Net output, in the case of a network device (e.g., switch or router), might include, for example, the number of calls handled per second. Redundancy adds to gross hardware capability but not to the net output and, thus, decreases service density. Adding hardware increases the likelihood of a failure and, thus, decreases the mean time between failures. Likewise, hot backup comes at the expense of system power. Each active element consumes some amount of the limited power available to the system. In general, the 1+1 or 1:1 redundancy designs provide the highest reliability but at a relatively high cost. Due to the importance of network availability, most network service providers prefer the 1+1 redundancy design to minimize network downtime.

In a 1:N redundancy design, instead of having one backup element per primary element, a single backup element or spare is used to backup multiple (N) primary elements. As a result, the 1:N design is generally less expensive to manufacture, offers greater service density and better mean time between failures than the 1:1 design and requires a smaller chassis / less space than a 1:1 design. One disadvantage of such a system, however, is that once a primary element fails over to the backup element, the system is no longer redundant (i.e., no available backup element for any primary element). Another disadvantage relates to hot state backup. Because one

backup element must support multiple primary elements, the typical 1:N design provides no hot state on the backup element leading to long synchronization times and, for network devices, the likelihood that connections will be dropped and availability reduced.

Even where the backup element provides some level of hot state backup it generally lacks the processing power and memory to provide a full hot state backup (i.e., 1+N) for all primary elements. To enable some level of hot state backup for each primary element, the backup element is generally a "mega spare" equipped with a more powerful processor and additional memory. This requires customers to stock more hardware than in a design with identical backup and primary elements. For instance, users typically maintain extra hardware in the case of a failure. If a primary fails over to the backup, the failed primary may be replaced with a new primary. If the primary and backup elements are identical, then users need only stock that one type of board, that is, a failed backup is also replaced with the same hardware used to replace the failed primary. If they are different, then the user must stock each type of board, thereby increasing the user's cost.

Distributed Redundancy:

A distributed redundancy architecture spreads software backup (hot state) across multiple elements. Each element may provide software backup for one or more other elements. For software backup alone, therefore, the distributed redundancy architecture eliminates the need for hardware backup elements (i.e., spare hardware). Where hardware backup is also provided, spreading resource demands across multiple elements makes it possible to have significant (perhaps full) hot state backup without the need for a mega spare. Identical backup (spare) and primary hardware provides manufacturing advantages and customer inventory advantages. A distributed redundancy design is less expensive than many 1:1 designs and a distributed redundancy architecture also permits the location of the hardware backup element to float, that is, if a primary element fails over to the backup element, when the failed primary element is replaced, that new hardware may serve as the hardware backup.

Software Redundancy:

In its simplest form, a distributed redundancy system provides software redundancy (i.e., backup) with or without redundant (i.e., backup) hardware, for example, with or without using backup line card 16n as discussed earlier with reference to the logical to physical card table (Fig. 14b). Referring to Fig. 29, computer system 10 includes primary line cards 16a, 16b and 16c. Computer system 10 will likely include additional primary line cards; only three are discussed herein (and shown in Fig. 29) for convenience. As described above, to load instances of software applications, the NMS creates software load records (SLR) 128a-128n in configuration database 42. The SLR includes the name of a control shim executable file and a logical identification (LID) associated with a primary line card on which the application is to be spawned. In the current example, there either are no hardware backup line cards or, if there are, the slave SRM executing on that line card does not download and execute backup applications.

As one example, NMS 60 creates SLR 128a including the executable name atm_cntrl.exe and card LID 30 (line card 16a), SLR 128b including atm_cntrl.exe and LID 31 (line card 16b) and SLR 128c including atm_cntrl.exe and LID 32 (line card 16c). The configuration database detects LID 30, 31 and 32 in SLRs 128a, 128b and 128c, respectively, and sends slave SRMs 37b, 37c and 37d (line cards 16a, 16b, and 16c) notifications including the name of the executable file (e.g., atm_cntrl.exe) to be loaded. The slave SRMs then download and execute a copy of atm_cntrl.exe 135 from memory 40 to spawn ATM controllers 136a, 136b and 136c.

Through the active query feature, the ATM controllers are sent records from group table (GT) 108' (Fig. 30) indicating how many instances of ATM each must start on their associated line cards. Group table 108' includes a primary line card LID field 447 and a backup line card LID field 449 such that, in addition to starting primary instances of ATM, each primary line card also executes backup instances of ATM. For example, ATM controller 136a receives records 450-453 and 458-461 from group table 108' including LID 30 (line card 16a). Records 450-453 indicate that ATM controller 136a is to start four primary instantiations of ATM 464-467 (Fig. 29), and records 458-461 indicate that ATM controller 136a is to start four backup instantiations of ATM 468-471 as backup for four primary instantiations on LID 32 (line card 16c). Similarly, ATM controller 136b receives records 450-457 from group

table 108' including LID 31 (line card 16b). Records 454-457 indicate that ATM controller 136b is to start four primary instantiations of ATM 472-475, and records 450-453 indicate that ATM controller 136b is to start four backup instantiations of ATM 476-479 as backup for four primary instantiations on LID 30 (line card 16a). ATM controller 136c receives records 454-461 from group table 108' including LID 32 (line card 16c). Records 458-461 indicate that ATM controller 136c is to start four primary instantiations of ATM 480-483, and records 454-457 indicate that ATM controller 136c is to start four backup instantiations of ATM 484-487 as backup for four primary instantiations on LID 31 (line card 16b). ATM controllers 136a, 136b and 136c then download atm.exe 138 and generate the appropriate number of ATM instantiations and also indicate to each instantiation whether it is a primary or backup instantiation. Alternatively, the ATM controllers may download atm.exe and generate the appropriate number of primary ATM instantiations and download a separate backup_atm.exe and generate the appropriate number of backup ATM instantiations.

Each primary instantiation registers with its local name server 220b-220d, as described above, and each backup instantiation subscribes to its local name server 220b-220d for information about its corresponding primary instantiation. The name server passes each backup instantiation at least the process identification number assigned to its corresponding primary instantiation, and with this, the backup instantiation sends a message to the primary instantiation to set up a dynamic state check-pointing procedure. Periodically or asynchronously as state changes, the primary instantiation passes dynamic state information to the backup instantiation (i.e., check-pointing). In one embodiment, a Redundancy Manager Service available from Harris and Jefferies of Dedham, Massachusetts may be used to allow backup and primary instantiations to pass dynamic state information. If the primary instantiation fails, it can be re-started, retrieve its last known dynamic state from the backup instantiation and then initiate an audit procedure (as described above) to resynchronize with other processes. The retrieval and audit process will normally be completed very quickly, resulting in no discernable service disruption.

Although each line card in the example above is instructed by the group table to start four instantiations of ATM, this is by way of example only. The user could instruct

the NMS to set up the group table to have each line card start one or more instantiations and to have each line card start a different number of instantiations.

Referring to Fig. 31a-31c, if one or more of the primary processes on element 16a (ATM 464-467) experiences a software fault (Fig. 31b), the processor on line card 16a may terminate and restart the failing process or processes. Once the process or processes are restarted (ATM 464'-467', Fig. 31c), they retrieve a copy of the last known dynamic state (i.e., backup state) from corresponding backup processes (ATM 476-479) executing on line card 16b and initiate an audit process to synchronize retrieved state with the dynamic state of associated other processes. The backup state represents the last known active or dynamic state of the process or processes prior to termination, and retrieving this state from line card 16b allows the restarted processes on line card 16a to quickly resynchronize and continue operating. The retrieval and audit process will normally be completed very quickly, and in the case of a network device, quick resynchronization may avoid losing network connections, resulting in no discernable service disruption.

If, instead of restarting a particular application, the software fault experienced by line card 16a requires the entire element to be shut down and rebooted, then all of the processes executing on line card 16a will be terminated including backup processes ATM 468-471. When the primary processes are restarted, backup state information is retrieved from backup processes executing on line card 16b as explained above. Simultaneously, the restarted backup processes on line card 16a again initiate the check-pointing procedure with primary ATM processes 480-483 executing on line card 16c to again serve as backup processes for these primary processes. Referring to Figs. 32a-32c, the primary processes executing on one line card may be backed-up by backup processes running on one or more other line cards. In addition, each primary process may be backed-up by one or more backup processes executing on one or more of the other line cards.

Since the operating system assigns each process its own memory block, each primary process may be backed-up by a backup process running on the same line card. This would minimize the time required to retrieve backup state and resynchronize if a primary process fails and is restarted. In a computer system that includes a spare or

backup line card (described below), the backup state is best saved on another line card such that in the event of a hardware fault, the backup state is not lost and can be copied from the other line card. If memory and processor limitations permit, backup processes may run simultaneously on the same line card as the primary process and on another line card such that software faults are recovered from using local backup state and hardware faults are recovered from using remote backup state.

Where limitations on processing power or memory make full hot state backup impossible or impractical, only certain hot state data will be stored as backup. The level of hot state backup is inversely proportional to the resynchronization time, that is, as the level of hot state backup increases, resynchronization time decreases. For a network device, backup state may include critical information that allows the primary process to quickly re-synchronize.

Critical information for a network device may include connection data relevant to established network connections (e.g., call set up information and virtual circuit information). For example, after primary ATM applications 464-467, executing on line card 16a, establish network connections, those applications send critical state information relevant to those connections to backup ATM applications 479-476 executing on line card 16b. Retrieving connection data allows the hardware (i.e., line card 16a) to send and receive network data over the previously established network connections preventing these connections from being terminated / dropped.

Although ATM applications were used in the examples above, this is by way of example only. Any application (e.g., IP or MPLS), process (e.g., MCD or NS) or device driver (e.g., port driver) may have a backup process started on another line card to store backup state through a check-pointing procedure.

Hardware and Software Backup:

By adding one or more hardware backup elements (e.g., line card 16n) to the computer system, the distributed redundancy architecture provides both hardware and software backup. Software backup may be spread across all of the line cards or only some of the line cards. For example, software backup may be spread only across the

primary line cards, only on one or more backup line cards or on a combination of both primary and backup line cards.

Referring to Fig. 33a, in the continuing example, line cards 16a, 16b and 16c are primary hardware elements and line card 16n is a spare or backup hardware element. In this example, software backup is spread across only the primary line cards. Alternatively, backup line card 16n may also execute backup processes to provide software backup. Backup line card 16n may execute all backup processes such that the primary elements need not execute any backup processes or line card 16n may execute only some of the backup processes. Regardless of whether backup line card 16n executes any backup processes, it is preferred that line card 16n be at least partially operational and ready to use the backup processes to quickly begin performing as if it was a failed primary line card.

There are many levels at which a backup line card may be partially operational. For example, the backup line card's hardware may be configured and device driver processes 490 loaded and ready to execute. In addition, the active state of the device drivers 492, 494, and 496 on each of the primary line cards may be stored as backup device driver state (DDS) 498, 500, 502 on backup line card 16n such that after a primary line card fails, the backup device driver state corresponding to that primary element is used by device driver processes 490 to quickly synchronize the hardware on backup line card 16n. In addition, data reflecting the network connections established by each primary process may be stored within each of the backup processes or independently on backup line card 16n, for example, connection data (CD) 504, 506, 508. Having a copy of the connection data on the backup line card allows the hardware to quickly begin transmitting network data over previously established connections to avoid the loss of these connections and minimize service disruption. The more operational (i.e., hotter) backup line card 16n is the faster it will be able to transfer data over network connections previously established by the failed primary line card and resynchronize with the rest of the system.

In the case of a primary line card hardware fault, the backup or spare line card takes the place of the failed primary line card. The backup line card starts new primary processes that register with the name server on the backup line card and begin

retrieving active state from backup processes associated with the original primary processes. As described above, the same may also be true for software faults. Referring to Fig. 33b, if, for example, line card 16a in computer system 10 is affected by a fault, the slave SRM executing on backup line card 16n may start new primary processes 464'-467' corresponding to the original primary processes 464-467. The new primary processes register with the name server process executing on line card 16n and begin retrieving active state from backup processes 476-479 on line card 16b. This is referred to as a "fail-over" from failed primary line card 16a to backup line card 16n.

As discussed above, preferably, backup line card 16n is partially operational. While active state is being retrieved from backup processes on line card 16b, device driver processes 490 use device driver state 502 and connection data 508 corresponding to failed primary line card 16a to quickly continue passing network data over previously established connections. Once the active state is retrieved then the ATM applications resynchronize and may begin establishing new connections and tearing down old connections.

Floating Backup Element:

Referring to Fig. 33c, when the fault is detected on line card 16a, diagnostic tests may be run to determine if the error was caused by software or hardware. If the fault is a software error, then line card 16a may again be used as a primary line card. If the fault is a hardware error, then line card 16a is replaced with a new line card 16a' that is booted and configured and again ready to be used as a primary element. In one embodiment, once line card 16a or 16a' is ready to serve as a primary element, a fail-over is initiated from line card 16n to line card 16a or 16a' as described above, including starting new primary processes 464''-467'' and retrieving active state from primary processes 464'-467' on line card 16n (or backup processes 476-479 on line card 16b). Backup processes 468''-471'' are also started, and those backup processes initiate a check-pointing procedure with primary processes 480-483 on line card 16c. This fail-over may cause the same level of service interruption as an actual failure.

Instead of failing-over from line card 16n back to line card 16a or 16a' and risking further service disruption, line card 16a or 16a' may serve as the new backup line card

with line card 16n serving as the primary line card. If line cards 16b, 16c or 16n experience a fault, a fail-over to line card 16a is initiated as discussed above and the primary line card that failed (or a replacement of that line card) serves as the new backup line card. This is referred to as a "floating" backup element. Referring to Fig. 33d, if, for example, line card 16c experiences a fault, primary processes 480'-483' are started on backup line card 16a and active state is retrieved from backup processes 464'-467' on line card 16n. After line card 16c is rebooted or replaced and rebooted, it serves as the new backup line card for primary line cards 16a, 16b and 16n.

Alternatively, computer system 10 may be physically configured to only allow a line card in a particular chassis slot, for example, line card 16n, to serve as the backup line card. This may be the case where physically, the slot line card 16n is inserted within is wired to provide the necessary connections to allow line card 16n to communicate with each of the other line cards but no other slot provides these connections. In addition, even where the computer system is capable of allowing line cards in other chassis slots to act as the backup line card, the person acting as network manager, may prefer to have the backup line card in each of his computer systems in the same slot. In either case, where only line card 16n serves as the backup line card, once line card 16a (or any other failed primary line card) is ready to act as a primary line card again, a fail-over, as described above, is initiated from line card 16n to the primary line card to allow line card 16n to again serve as a backup line card to each of the primary line cards.

Balancing Resources:

Typically, multiple processes or applications are executed on each primary line card. Referring to Fig. 34a, in one embodiment, each primary line card 16a, 16b, 16c executes four applications. Due to physical limitations (e.g., memory space, processor power), each primary line card may not be capable of fully backing up four applications executing on another primary line card. The distributed redundancy architecture allows backup processes to be spread across multiple line cards, including any backup line cards, to more efficiently use all system resources.

For instance, primary line card 16a executes backup processes 510 and 512 corresponding to primary processes 474 and 475 executing on primary line card 16b.

Primary line card 16b executes backup processes 514 and 516 corresponding to primary processes 482 and 483 executing on primary line card 16c, and primary line card 16c executes backup processes 518 and 520 corresponding to primary processes 466 and 467 executing on primary line card 16a. Backup line card 16n executes backup processes 520, 522, 524, 526, 528 and 530 corresponding to primary processes 464, 465, 472, 473, 480 and 481 executing on each of the primary line cards. Having each primary line card execute backup processes for only two primary processes executing on another primary line card reduces the primary line card resources required for backup. Since backup line card 16n is not executing primary processes, more resources are available for backup. Hence, backup line card 16n executes six backup processes corresponding to six primary processes executing on primary line cards. In addition, backup line card 16n is partially operational and is executing device driver processes 490 and storing device driver backup state 498, 500 and 502 corresponding to the device drivers on each of the primary elements and network connection data 504, 506 and 508 corresponding to the network connections established by each of the primary line cards.

Alternatively, each primary line card could execute more or less than two backup processes. Similarly, each primary line card could execute no backup processes and backup line card 16n could execute all backup processes. Many alternatives are possible and backup processes need not be spread evenly across all primary line cards or all primary line cards and the backup line card.

Referring to Fig. 34b, if primary line card 16b experiences a failure, device drivers 490 on backup line card 16n begins using the device driver state, for example, DDS 498, corresponding to the device drivers on primary line card 16b and the network connection data, for example, CD 506, corresponding to the connections established by primary line card 16b to continue transferring network data. Simultaneously, backup line card 16n starts substitute primary processes 510' and 512' corresponding to the primary processes 474 and 475 on failed primary line card 16b. Substitute primary processes 510' and 512' retrieve active state from backup processes 510 and 512 executing on primary line card 16a. In addition, the slave SRM on backup line card 16n informs backup processes 526 and 524 corresponding to primary processes 472 and 473 on failed primary line card 16b that they are now primary processes. The

new primary applications then synchronize with the rest of the system such that new network connections may be established and old network connections torn down. That is, backup line card 16n begins operating as if it were primary line card 16b.

Multiple Backup Elements:

In the examples given above, one backup line card is shown. Alternatively, multiple backup line cards may be provided in a computer system. In one embodiment, a computer system includes multiple different primary line cards. For example, some primary line cards may support the Asynchronous Transfer Mode (ATM) protocol while others support the Multi-Protocol Label Switching (MPLS) protocol, and one backup line card may be provided for the ATM primary line cards and another backup line card may be provided for the MPLS primary line cards. As another example, some primary line cards may support four ports while others support eight ports and one backup line card may be provided for the four port primaries and another backup line card may be provided for the eight port primaries. One or more backup line cards may be provided for each different type of primary line card.

Data Plane:

Referring to Fig. 35, a network device 540 includes a central processor 542, a redundant central processor 543 and a Fast Ethernet control bus 544 similar to central processors 12 and 13 and Ethernet 32 discussed above with respect to computer system 10. In addition, network device 540 includes forwarding cards (FC) 546a-546e, 548a-548e, 550a-550e and 552a-552e that are similar to line cards 16a-16n discussed above with respect to computer system 10. Network device 540 also includes (and computer system 10 may also include) universal port (UP) cards 554a-554h, 556a-556h, 558a-558h, and 560a-560h, cross-connection (XC) cards 562a-562b, 564a-564b, 566a-566b, and 568a-568b, and switch fabric (SF) cards 570a-570b. In one embodiment, network device 540 includes four quadrants where each quadrant includes five forwarding cards (e.g., 546a-546e), two cross connection cards (e.g., 562a-562b) and eight universal port cards (e.g., 554a-554h). Network device 540 is a distributed processing system. Each of the cards includes a processor and is connected to the Ethernet control bus. In addition, each of the cards are configured as described above with respect to line cards.

In one embodiment, the forwarding cards have a 1:4 hardware redundancy structure and distributed software redundancy as described above. For example, forwarding card 546e is the hardware backup for primary forwarding cards 546a-546d and each of the forwarding cards provide software backup. The cross-connection cards are 1:1 redundant. For example, cross-connection card 562b provides both hardware and software backup for cross-connection card 562a. Each port on the universal port cards may be 1:1, 1+1, 1:N redundant or not redundant at all depending upon the quality of service paid for by the customer associated with that port. For example, port cards 554e-554h may be the hardware and software backup cards for port cards 554a-554d in which case the port cards are 1:1 or 1+1 redundant. As another example, one or more ports on port card 554a may be backed-up by separate ports on one or more port cards (e.g., port cards 554b and 554c) such that each port is 1:1 or 1+1 redundant, one or more ports on port card 554a may not be backed-up at all (i.e., not redundant) and two or more ports on 554a may be backed-up by one port on another port card (e.g., port card 554b) such that those ports are 1:N redundant. Many redundancy structures are possible using the LID to PID Card table (LPCT) 100 (Fig. 14b) and LID to PID Port table (LPPT) as described above.

Each port card includes one or more ports for connecting to external network connections. One type of network connection is an optical fiber carrying an OC-48 SONET stream, and as described above, an OC-48 SONET stream may include connections to one or more end points using one or more paths. A SONET fiber carries a time division multiplexed (TDM) byte stream of aggregated time slots (TS). A time slot has a bandwidth of 51 Mbps and is the fundamental unit of bandwidth for SONET. An STS-1 path has one time slot within the byte stream dedicated to it, while an STS-3c path (i.e., three concatenated STS-1s) has three time slots within the byte stream dedicated to it. The same or different protocols may be carried over different paths within the same TDM byte stream. In other words, ATM over SONET may be carried on an STS-1 path within a TDM byte stream that also includes IP over SONET on another STS-1 path or on an STS-3c path.

Through network management system 60 on workstation 62, after a user connects an external network connection to a port, the user may enable that port and one or more paths within that port (described below). Data received on a port card path is passed

to the cross-connection card in the same quadrant as the port card, and the cross-connection card passes the path data to one of the five forwarding cards or eight port cards also within the same quadrant. The forwarding card determines whether the payload (e.g., packets, frames or cells) it is receiving includes user payload data or network control information. The forwarding card itself processes certain network control information and sends certain other network control information to the central processor over the Fast Ethernet control bus. The forwarding card also generates network control payloads and receives network control payloads from the central processor. The forwarding card sends any user data payloads from the cross-connection card or control information from itself or the central processor as path data to the switch fabric card. The switch fabric card then passes the path data to one of the forwarding cards in any quadrant, including the forwarding card that just sent the data to the switch fabric card. That forwarding card then sends the path data to the cross-connection card within its quadrant, which passes the path data to one of the port cards within its quadrant.

Referring to Fig. 36, in one embodiment, a universal port card 554a includes one or more ports 571a-571n connected to one or more transceivers 572a-572n. The user may connect an external network connection to each port. As one example, port 571a is connected to an ingress optical fiber 576a carrying an OC-48 SONET stream and an egress optical fiber 576b carrying an OC-48 SONET stream. Port 571a passes optical data from the SONET stream on fiber 576a to transceiver 572a. Transceiver 572a converts the optical data into electrical signals that it sends to a SONET framer 574a. The SONET framer organizes the data it receives from the transceiver into SONET frames. SONET framer 574a sends data over a telecommunications bus 578a to a serializer-deserializer (SERDES) 580a that serializes the data into four serial lines with twelve STS-1 time slots each and transmits the four serial lines to cross-connect card 562a.

Each cross-connection card is a switch that provides connections between port cards and forwarding cards within its quadrant. Each cross-connection card is programmed to transfer each serial line on each port card within its quadrant to a forwarding card within its quadrant or to serial line on a port card, including the port card that

transmitted the data to the cross-connection card. The programming of the cross-connect card is discussed in more detail below under Policy Based Provisioning.

Each forwarding card (e.g., forwarding card 546c) receives SONET frames over serial lines from the cross-connection card in its quadrant through a payload extractor chip (e.g., payload extractor 582a). In one embodiment, each forwarding card includes four payload extractor chips where each payload extractor chip represents a "slice" and each serial line input represents a forwarding card "port". Each payload extractor chip receives four serial line inputs, and since each serial line includes twelve STS-1 time slots, the payload extractor chips combine and separate time slots where necessary to output data paths with the appropriate number of time slots. Each STS-1 time slot may represent a separate data path, or multiple STS-1 time slots may need to be combined to form a data path. For example, an STS-3c path requires the combination of three STS-1 time slots to form a data path while an STS-48c path requires the combination of all forty-eight STS-1 time slots. Each path represents a separate network connection, for example, an ATM cell stream.

The payload extractor chip also strips off all vestigial SONET frame information and transfers the data path to an ingress interface chip. The ingress interface chip will be specific to the protocol of the data within the path. As one example, the data may be formatted in accordance with the ATM protocol and the ingress interface chip is an ATM interface chip (e.g., ATM IF 584a). Other protocols can also be implemented including, for example, Internet Protocol (IP), Multi-Protocol Label Switching (MPLS) protocol or Frame Relay.

The ingress ATM IF chip performs many functions including determining connection information (e.g., virtual circuit or virtual path information) from the ATM header in the payload. The ATM IF chip uses the connection information as well as a forwarding table to perform an address translation from the external address to an internal address. The ATM IF chip passes ATM cells to an ingress bridge chip (e.g., BG 586a-586b) which serves as an interface to an ingress traffic management chip or chip set (e.g., TM 588a-588n).

The traffic management chips ensure that high priority traffic, for example, voice data, is passed to switch fabric card 570a faster than lower priority traffic, for example, e-mail data. The traffic management chips may buffer lower priority traffic while higher priority traffic is transmitted, and in times of traffic congestion, the traffic management chips will ensure that low priority traffic is dropped prior to any high priority traffic. The traffic management chips also perform an address translation to add the address of the traffic management chip to which the data is going to be sent by the switch fabric card. The address corresponds to internal virtual circuits set up between forwarding cards by the software and available to the traffic management chips in tables.

The traffic management chips send the modified ATM cells to switch fabric interface chips (SFIF) 589a-589n that then transfer the ATM cells to switch fabric card 570a. The switch fabric card uses the address provided by the ingress traffic management chips to pass ATM cells to the appropriate egress traffic management chips (e.g., TM 590a-590n) on the various forwarding cards. In one embodiment, the switch fabric card 570a is a 320 Gbps, non-blocking fabric. Since each forwarding card serves as both an ingress and egress, the switching fabric card provides a high degree of flexibility in directing the data between any of the forwarding cards, including the forwarding card that sent the data to the switch fabric card.

When a forwarding card (e.g., forwarding card 546c) receives ATM cells from switch fabric card 570a, the egress traffic management chips re-translate the address of each cell and pass the cells to egress bridge chips (e.g., BG 592a-592b). The bridge chips pass the cells to egress ATM interface chips (e.g., ATM IF 594a-594n), and the ATM interface chips add a re-translated address to the payload representing an ATM virtual circuit. The ATM interface chips then send the data to the payload extractor chips (e.g., payload extractor 582a-582n) that separate, where necessary, the path data into STS-1 time slots and combine twelve STS-1 time slots into four serial lines and send the serial lines back through the cross-connection card to the appropriate port card.

The port card SERDES chips receive the serial lines from the cross-connection card and de-serialize the data and send it to SONET framer chips 574a-574n. The Framers properly format the SONET overhead and send the data back through the transceivers

that change the data from electrical to optical before sending it to the appropriate port and SONET fiber.

Although the port card ports above were described as connected to a SONET fiber carrying an OC-48 stream, other SONET fibers carrying other streams (e.g., OC-12) and other types of fibers and cables, for example, Ethernet, may be used instead. The transceivers are standard parts available from many companies, including Hewlett Packard Company and Sumitomo Corporation. The SONET framer may be a Spectra chip available from PMC-Sierra, Inc. in British Columbia. A Spectra 2488 has a maximum bandwidth of 2488 Mbps and may be coupled with a 1xOC48 transceiver coupled with a port connected to a SONET optical fiber carrying an OC-48 stream also having a maximum bandwidth of 2488 Mbps. Instead, four SONET optical fibers carrying OC-12 streams each having a maximum bandwidth of 622Mbps may be connected to four 1xOC12 transceivers and coupled with one Spectra 2488. Alternatively, a Spectra 4x155 may be coupled with four OC-3 transceivers that are coupled with ports connected to four SONET fibers carrying OC-3 streams each having a maximum bandwidth of 155 Mbps. Many variables are possible.

The SERDES chip may be a Telecommunications Bus Serializer (TBS) chip from PMC-Sierra, and each cross-connection card may include a Time Switch Element (TSE) from PMC-Sierra, Inc. Similarly, the payload extractor chips may be MACH 48 chips and the ATM interface chips may be ATLAS chips both of which are available from PMC-Sierra. Several chips are available from Extreme Packet Devices (EPD), a subsidiary of PMC-Sierra, including PP3 bridge chips and Data Path Element (DPE) traffic management chips. The switch fabric interface chips may include a Switch Fabric Interface (SIF) chip also from EPD. Other switch fabric interface chips are available from Abrizio, also a subsidiary of PMC-Sierra, including a data slice chip and an enhanced port processor (EPP) chip. The switch fabric card may also include chips from Abrizio, including a cross-bar chip and a scheduler chip.

Although the port cards, cross-connection cards and forwarding cards have been shown as separate cards, this is by way of example only and they may be combined into one or more different cards.

Multiple Redundancy Schemes:

Coupling universal port cards to forwarding cards through a cross-connection card provides flexibility in data transmission by allowing data to be transmitted from any path on any port to any port on any forwarding card. In addition, decoupling the universal port cards and the forwarding cards enables redundancy schemes (e.g., 1:1, 1+1, 1:N, no redundancy) to be set up separately for the forwarding cards and universal port cards. The same redundancy scheme may be set up for both or they may be different. As described above, the LID to PID card and port tables are used to setup the various redundancy schemes for the line cards (forwarding or universal port cards) and ports. Network devices often implement industry standard redundancy schemes, such as those defined by the Automatic Protection Switching (APS) standard. In network device 540 (Fig. 35), an APS standard redundancy scheme may be implemented for the universal port cards while another redundancy scheme is implemented for the forwarding cards.

Referring again to Fig. 35, further data transmission flexibility may be provided by connecting (i.e., connections 565) each cross-connection card 562a-562b, 564a-564b, 566a-566b and 568a-568b to each of the other cross-connection cards. Through connections 565, a cross-connection card (e.g., cross-connection card 562a) may transmit data between any port or any path on any port on a universal port card (e.g., universal port cards 554a-554h) in its quadrant to a cross-connection card (e.g., cross-connection card 568a) in any other quadrant, and that cross-connection card (e.g., cross-connection card 568a) may transmit the data to any forwarding card (e.g., forwarding cards 552a-552e) or universal port card (e.g., universal port cards 560a-560h) in its quadrant. Similarly, any cross-connection card may transmit data received from any forwarding card in its quadrant to any other cross-connection card and that cross-connection card may transmit the data to any universal port card port in its quadrant.

Alternatively, the cross-connection cards in each quadrant may be coupled only with cross-connection cards in one other quadrant. For example, cross-connection cards in quadrants 1 and 2 may be connected and cross-connection cards in quadrants 3 and 4 may be connected. Similarly, the cross-connection cards in each quadrant may be coupled with cross-connection cards in only two other quadrants, or only the cross-

connection cards in one quadrant (e.g., quadrant 1) may be connected to cross-connection cards in another quadrant (e.g., quadrant 2) while the cross-connection cards in the other quadrants (e.g., quadrants 3 and 4) are not connected to other cross-connection cards or are connected only to cross-connection cards in one quadrant (e.g., quadrant 2). Many variations are possible. Although these connections do not provide the flexibility of having all cross-connection cards inter-connected, these connections require less routing resources and still provide some increase in the data transmission flexibility of the network device.

The additional flexibility provided by inter-connecting one or more cross-connection cards may be used to optimize the efficiency of network device 540. For instance, a redundant forwarding card in one quadrant may be used as a backup for primary forwarding cards in other quadrants thereby reducing the number of backup modules and increasing the network device's service density. Similarly, a redundant universal port card or a redundant port on a universal port card in one quadrant may be used as a backup for primary universal port cards or ports in other quadrants. As previously mentioned, each primary forwarding card may support a different protocol (e.g., ATM, MPLS, IP, Frame Relay). Similarly, each universal port card may support a different protocol (e.g., SONET, Ethernet). A backup or spare forwarding card or universal port card must support the same protocol as the primary card or cards. If forwarding or universal port cards in one quadrant support multiple protocols and the cross-connection cards are not interconnected, then each quadrant may need multiple backup forwarding and universal port cards (i.e., one for each protocol supported). If each of the quadrants includes forwarding and universal port cards that support different protocols then each quadrant may include multiple backup forwarding and universal port cards further decreasing the network device's service density.

By inter-connecting the cross-connection cards, a forwarding card in one quadrant may serve as a backup for primary forwarding cards in its own quadrant and in other quadrants. Similarly, a universal port card or port in one quadrant may serve as a backup for a primary universal port card or port in its own quadrant and in other quadrants. For example, forwarding card 546e in quadrant 1 that supports a particular protocol (e.g., the ATM protocol) may serve as the backup forwarding card for primary forwarding cards supporting ATM in its own quadrant (e.g., forwarding cards

546a-546b) as well as for primary forwarding cards supporting ATM in quadrant 2 (e.g., forwarding cards 548b-548c) or all quadrants (e.g., forwarding card 550c in quadrant 3 and forwarding cards 552b-552d in quadrant 4). Similarly, forwarding card 548e in quadrant 2 that supports a different protocol (e.g., the MPLS protocol) may serve as the backup forwarding card for primary forwarding cards supporting MPLS in its own quadrant (e.g., forwarding cards 548a and 548d) as well as for primary forwarding cards supporting MPLS in quadrant 1 (e.g., forwarding card 546c) or all quadrants (e.g., forwarding card 550a in quadrant 3 and forwarding card 552a in quadrant 4). Even with this flexibility, to provide sufficient redundancy, multiple backup modules supporting the same protocol may be used, especially where a large number of primary modules support one protocol.

As previously discussed, each port on a universal port card may be connected to an external network connection, for example, an optical fiber transmitting data according to the SONET protocol. Each external network connection may provide multiple streams or paths and each stream or path may include data being transmitted according to a different protocol over SONET. For example, one path may include data being transmitted according to ATM over SONET while another path may include data being transmitted according to MPLS over SONET. The cross-connection cards may be programmed (as described below) to transmit protocol specific data (e.g., ATM, MPLS, IP, Frame Relay) from ports on universal port cards within their quadrants to forwarding cards within any quadrant that support the specific protocol. Because the traffic management chips on the forwarding cards provide protocol-independent addresses to be used by switch fabric cards 570a-570b, the switch fabric cards may transmit data between any of the forwarding cards regardless of the underlying protocol.

Alternatively, the network manager may dedicate each quadrant to a specific protocol by putting forwarding cards in each quadrant according to the protocol they support. Within each quadrant then, one forwarding card may be a backup card for each of the other forwarding cards (1:N, for network device 540, 1:4). Protocol specific data received from ports or paths on ports on universal port cards within any quadrant may then be forwarded by one or more cross-connection cards to forwarding cards within the protocol specific quadrant. For instance, quadrant 1 may include forwarding cards

for processing data transmissions using the ATM protocol, quadrant 2 may include forwarding cards for processing data transmissions using the IP protocol, quadrant 3 may include forwarding cards for processing data transmissions using the MPLS protocol and quadrant 4 may be used for processing data transmissions using the Frame Relay protocol. ATM data received on a port path is then transmitted by one or more cross-connection cards to a forwarding card in quadrant 1, while MPLS data received on another path on that same port or on a path in another port is transmitted by one or more cross-connection cards to a forwarding card in quadrant 3.

Policy Based Provisioning:

Unlike the switch fabric card, the cross-connection card does not examine header information in a payload to determine where to send the data. Instead, the cross-connection card is programmed to transmit payloads, for example, SONET frames, between a particular serial line on a universal port card port and a particular serial line on a forwarding card port regardless of the information in the payload. As a result, one port card serial line and one forwarding card serial line will transmit data to each other through the cross-connection card until that programmed connection is changed.

In one embodiment, connections established through a path table and service endpoint table (SET) in a configuration database are passed to path managers on port cards and service endpoint managers (SEMs) on forwarding cards, respectively. The path managers and service endpoint managers then communicate with a cross-connect manager (CCM) on the cross-connection card in their quadrant to provide connection information. The CCM uses the connection information to generate a connection program table that is used by one or more components (e.g., a TSE chip 563) to program internal connection paths through the cross-connection card.

Typically, connections are fixed or are generated according to a predetermined map with a fixed set of rules. Unfortunately, a fixed set of rules may not provide flexibility for future network device changes or the different needs of different users / customers. Instead, within network device 540, each time a user wishes to enable / configure a path on a port on a universal port card, a Policy Provisioning Manager (PPM) 599 (Fig. 37) executing on central processor 542 selects the forwarding card port to which the port card port will be connected based on a configurable

provisioning policy (PP) 603 in configuration database 42. The configurable provisioning policy may take into consideration many factors such as available system resources, balancing those resources and quality of service. Similar to other programs and files stored within the configuration database of computer system 10 described above, the provisioning policy may be modified while network device 540 is running to allow to policy to be changed according to a user's changing needs or changing network device system requirements.

When a user connects an external network connection to a particular port on a universal port card, the user notifies the NMS as to which port on which universal port card should be enabled, which path or paths should be enabled, and the number of time slots in each path. The user may also notify the NMS as to a new path and its number of time slots on an already enabled port that was not fully utilized or the user may notify the NMS of a modification to one or more paths on already enabled ports and the number of time slots required for that path or paths. With this information, the NMS fills in a Path table 600 (Figs. 37 and 38) and partially fills in a Service Endpoint Table (SET) 76' (Figs. 37 and 39).

When a record in the path table is filled in, the configuration database sends an active query notification to a path manager (e.g., path manager 597) executing on a universal port card (e.g., port card 554a) corresponding to the universal port card port LID (e.g., port 1231, Fig. 38) in the path table record (e.g., record 602).

Leaving some fields in the SET blank or assigning a particular value (e.g., zero), causes the configuration database to send an active query notification to Policy Provisioning Manager (PPM) 599. The PPM then determines – using provisioning policy 603 -- which forwarding card (FC) port or ports to assign to the new path or paths. For example, the PPM may first compare the new path's requirements, including its protocol (e.g., ATM over SONET), the number of time slots, the number of virtual circuits and virtual circuit scheduling restrictions, to the available forwarding card resources in the quadrant containing the universal port card port and path. The PPM also takes other factors into consideration including quality of service, for example, redundancy requirements or dedicated resource requirements, and balancing resource usage (i.e., load balancing) evenly within a quadrant.

As an example, a user connects SONET optical fiber 576a (Fig. 36) to port 571a on universal port card 554a and wants to enable a path with three time slots (i.e., STS-3c). The NMS assigns a path LID number (e.g., path LID 1666) and fills in a record (e.g., row 602) in Path Table 600 to include path LID 1666, a universal port card port LID (e.g., UP port LID 1231) previously assigned by the NMS and retrieved from the Logical to Physical Port Table, the first time slot (e.g., time slot 4) in the SONET stream corresponding with the path and the total number of time slots – in this example, 3 -- in the path. Other information may also be filled into Path Table 600.

The NMS also partially fills in a record (e.g., row 604) in SET 76' by filling in the quadrant number – in this example, 1 – and the assigned path LID 1666 and by assigning a service endpoint number 878. The SET table also includes other fields, for example, a forwarding card LID field 606, a forwarding card slice 608 (i.e., port) and a forwarding card serial line 610. In one embodiment, the NMS fills in these fields with a particular value (e.g., zero), and in another embodiment, the NMS leaves these fields blank.

In either case, the particular value or a blank field causes the configuration database to send an active query notice to the PPM indicating a new path LID, quadrant number and service endpoint number. It is up to the PPM to decide which forwarding card, slice (i.e., payload extractor chip) and time slot (i.e., port) to assign to the new universal port card path. Once decided, the PPM fills in the SET Table fields. Since the user and NMS do not completely fill in the SET record, this may be referred to as a "self-completing configuration record." Self-completing configuration records reduce the administrative workload of provisioning a network.

The SET and path table records may be automatically copied to persistent storage 21 to insure that if network device 540 is re-booted these configuration records are maintained. If the network device shuts down prior to the PPM filling in the SET record fields and having those fields saved in persistent storage, when the network device is rebooted, the SET will still include blank fields or fields with particular values which will cause the configuration database to again send an active query to the PPM.

When the forwarding card LID (e.g., 1667) corresponding, for example, to forwarding card 546c, is filled into the SET table, the configuration database sends an active query notification to an SEM (e.g., SEM 96i) executing on that forwarding card and corresponding to the assigned slice and/or time slots. The active query notifies the SEM of the newly assigned service endpoint number (e.g., SE 878) and the forwarding card slice (e.g., payload extractor 582a) and time slots (i.e., 3 time slots from one of the serial line inputs to payload extractor 582a) dedicated to the new path.

Path manager 597 and SEM 96i both send connection information to a cross-connection manager 605 executing on cross-connection card 562a – the cross-connection card within their quadrant. The CCM uses the connection information to generate a connection program table 601 and uses this table to program internal connections through one or more components (e.g., a TSE chip 563) on the cross-connection card. Once programmed, cross-connection card 562a transmits data between new path LID 1666 on SONET fiber 576a connected to port 571a on universal port card 554a and the serial line input to payload extractor 582a on forwarding card 546c.

An active query notification is also sent to NMS database 61, and the NMS then displays the new system configuration to the user.

Alternatively, the user may choose which forwarding card to assign to the new path and notify the NMS. The NMS would then fill in the forwarding card LID in the SET, and the PPM would only determine which time slots and slice within the forwarding card to assign.

In the description above, when the PPM is notified of a new path, it compares the requirements of the new path to the available / unused forwarding card resources. If the necessary resources are not available, the PPM may signal an error. Alternatively, the PPM could move existing forwarding card resources to make the necessary forwarding card resources available for the new path. For example, if no payload extractor chip is completely available in the entire quadrant, one path requiring only one time slot is assigned to payload extractor chip 582a and a new path requires forty-

eight time slots, the one path assigned to payload extractor chip 582a may be moved to another payload extractor chip, for example, payload extractor chip 582b that has at least one time slot available and the new path may be assigned all of the time slots on payload extractor chip 582a. Moving the existing path is accomplished by having the PPM modify an existing SET record. The new path is configured as described above.

Moving existing paths may result in some service disruption. To avoid this, the provisioning policy may include certain guidelines to hypothesize about future growth. For example, the policy may require small paths – for example, three or less time slots – to be assigned to payload extractor chips that already have some paths assigned instead of to completely unassigned payload extractor chips to provide a higher likelihood that forwarding card resources will be available for large paths – for example, sixteen or more time slots -- added in the future.

Multi-Layer Network Device in One Telco Rack:

Referring again to Fig. 35, in one embodiment, each universal port card includes four ports, each of which is capable of being connected to an OC-48 SONET fiber. Since an OC-48 SONET fiber is capable of transferring data at 2.5 Giga bits per second (Gbps), each universal port card is capable of transferring data at 10 Gbps (4x2.5=10). With eight port cards per quadrant, the cross-connection card must be capable of transferring data at 80 Gbps. Typically, however, the eight port cards will be 1:1 redundant and only transfer 40 Gbps. In one embodiment, each forwarding card is capable of transferring 10 Gbps, and with five forwarding cards per quadrant, the switch fabric cards must be capable of transferring data at 200 Gbps. Typically, however, the five forwarding cards will be 1:N redundant and only transfer data at 40 Gbps. With four quadrants and full redundancy (1:1 for port cards and 1:N for forwarding cards), network device 540 is capable of transferring data at 160 Gbps.

In other embodiments, each port card includes one port capable of being connected to an OC-192 SONET fiber. Since OC-192 SONET fibers are capable of transferring data at 10 Gbps, a fully redundant network device 540 is again capable of transferring 160 Gbps. In the embodiment employing one OC-192 connection per port card, each port card may include one hundred and ninety-two logical DS3 connections using sub-rate data multiplexing (SDRM). In addition, each port card may differ in its

number and type of ports to provide more or less data through put. As previously mentioned, ports other than SONET ports may be provided, for example, Ethernet ports, Plesiochronous Digital Hierarchy ports (i.e., DS0, DS1, DS3, E0, E1, E3, J0, J1, J3) and Synchronous Digital Hierarchy (SDH) ports (i.e., STM1, STM4, STM16, STM64).

The universal port cards and cross-connect cards in each quadrant are in effect a physical layer switch, and the forwarding cards and switch fabric cards are effectively an upper layer switch. Prior systems have packaged these two switches into separate network devices. One reason for this is the large number of signals that need to be routed. Taken separately, each cross-connect card 562a-562b, 564a-564b, 566a-566b and 568a-568b is essentially a switch fabric or mesh allowing switching between any path on any universal port card to any serial input line on any forwarding card in its quadrant and each switch fabric card 570a-570b allows switching between any paths on any forwarding cards. Approximately six thousand, seven hundred and twenty etches are required to support a 200 Gbps switch fabric, and about eight hundred and thirty-two etches are required to support an 80 Gbps cross-connect. Combining such high capacity multi-layer switches into one network device in a single telco rack (seven feet by nineteen inches by 24 inches) has not been thought possible by those skilled in the art of telecommunications network devices.

To fit network device 540 into a single telco rack, dual mid-planes are used. All of the functional printed circuit boards connect to at least one of the mid-planes, and the switch fabric cards and certain control cards connect to both mid-planes thereby providing connections between the two mid-planes. In addition, to efficiently utilize routing resources, instead of providing a single cross-connection card, the cross-connection functionality is separated into four cross-connection cards – one for each quadrant – (as shown in Fig. 35). Further, routing through the lower mid-plane is improved by flipping the forwarding cards and cross-connection cards in the bottom half of the front of the chassis upside down to be the mirror image of the forwarding cards and cross-connection cards in the top of the front half of the chassis.

Referring to Fig. 40, a network device 540 is packaged in a box 619 conforming to the telco standard rack of seven feet in height, nineteen inches in width and 24 inches in

depth. Referring also to Figs. 41a-41c, a chassis 620 within box 619 provides support for forwarding cards 546a-546e, 548a-548e, 550a-550e and 552a-552e, universal port cards 554a-554h, 556a-556h, 558a-558h and 560a-560h, and cross-connection cards 562a-562b, 564a-564b, 566a-566b and 568a-568b. As is typical of telco network devices, the forwarding cards (FC) are located in the front portion of the chassis where network administrators may easily add and remove these cards from the box, and the universal port cards (UP) are located in the back portion of the chassis where external network attachments / cables may be easily connected.

The chassis also supports switch fabric cards 570a and 570b. As shown, each switch fabric card may include multiple switch fabric (SF) cards and a switch scheduler (SS) card. In addition, the chassis supports multiple central processor cards (542 and 543, Fig. 35). Instead of having a single central processor card, the external control functions and the internal control functions may be separated onto different cards as described in U.S. Patent Application Serial Number ___, filed May 20, 2000 and entitled "Functional Separation of Internal and External Controls in Network Devices", which is hereby incorporated herein by reference. As shown, the chassis may support internal control (IC) processor cards 542a and 543a and external control (EC) processor cards 542b and 543b. Auxiliary processor (AP) cards 542c and 543c are provided for future expansion to allow more external control cards to be added, for example, to handle new upper layer protocols. In addition, a management interface (MI) card 621 for connecting to an external network management system (62, Fig. 35) is also provided.

The chassis also support two mid-plane printed circuit boards 622a and 622b (Fig. 41c) located toward the middle of chassis 620. Mid-plane 622a is located in the top portion of chassis 620 and is connected to quadrant 1 and 2 forwarding cards 546a-546e and 548a-548e, universal port cards 554a-554h and 556a-556h, and cross-connection cards 562a-562b and 564a-564b. Similarly, mid-plane 622b is located in the bottom portion of chassis 620 and is connected to quadrant 3 and 4 forwarding cards 550a-550e and 552a-552e, universal port cards 558a-558h and 560a-560h, and cross-connection cards 566a-566b and 568a-568b. Through each mid-plane, the cross-connection card in each quadrant may transfer network packets between any of the universal port cards in its quadrant and any of the forwarding cards in its quadrant.

In addition, through mid-plane 622a the cross-connection cards in quadrants 1 and 2 may be connected to allow for transfer of network packets between any forwarding cards and port cards in quadrants 1 and 2, and through mid-plane 622b the cross-connection cards in quadrants 3 and 4 may be connected to allow for transfer of network packets between any forwarding cards and port cards in quadrants 3 and 4.

Mid-plane 622a is also connected to external control processor cards 542b and 543b and management interface card 621. Mid-plane 622b is also connected to auxiliary processor cards 542c and 543c.

Switch fabric cards 570a and 570b are located in the back portion of chassis 620, approximately mid-way between the top and bottom of the chassis. The switch fabric cards are connected to both mid-planes 622a and 622b to allow the switch fabric cards to transfer signals between any of the forwarding cards in any quadrant. In addition, the cross-connection cards in quadrants 1 and 2 may be connected through the mid-planes and switch fabric cards to the cross-connection cards in quadrants 3 and 4 to enable network packets to be transferred between any universal port card and any forwarding card.

To provide for better routing efficiency through mid-plane 622b, forwarding cards 550a-550e and 552a-552e and cross-connection cards 566a-566b and 568a-568b in quadrants 3 and 4, located in the bottom portion of the chassis, are flipped over when plugged into mid-plane 622b. This permits the switch fabric interface 589a-589n on each of the lower forwarding cards to be oriented nearest the switch fabric cards and the cross-connection interface 582a-582n on each of the lower forwarding cards to be oriented nearest the cross-connection cards in quadrants 3 and 4. This orientation avoids having to cross switch fabric and cross-connection etches in mid-plane 622b.

Typically, airflow for cooling a network device is brought in at the bottom of the device and released at the top of the device. For example, in the back portion of chassis 620, a fan tray (FT) 626 pulls air into the device from the bottom portion of the device and a fan tray 628 blows air out of the top portion of the device. When the lower forwarding cards are flipped over, the airflow / cooling pattern is reversed. To accommodate this reversal, fan trays 630 and 632 pull air into the middle portion of

the device and then fan trays 634 and 636 pull the air upwards and downwards, respectively, and blow the heated air out the top and bottom of the device, respectively.

The quadrant 3 and 4 universal port cards 558a-558h and 560a-560h may also be flipped over to orient the port card's cross-connection interface nearest the cross-connection cards and more efficiently use the routing resources. It is preferred, however, not to flip the universal port cards for serviceability reasons and airflow issues. The network managers at the telco site expect network attachments / cables to be in a certain pattern. Reversing this pattern could cause confusion in a large telco site with many different types of network devices. Also, flipping the port cards will change the airflow and cooling pattern and require a similar airflow pattern and fan tray configuration as implemented in the front of the chassis. However, with the switch fabric and internal control processor cards in the middle of the back portion of the chassis, it may be impossible to implement this fan tray configuration.

Referring to Fig. 42, mid-plane 622a includes connectors 638 mounted on the back side of the mid-plane ("back mounted") for the management interface card, connectors 640a-640d mounted on the front side of the mid-plane ("front mounted") for the quadrant 1 and 2 cross-connection cards, and front mounted connectors 642a-642b for the external control processor cards. Multiple connectors may be used for each card. Mid-plane 622a also includes back mounted connectors 644a-644p for the quadrant 1 and 2 universal port cards and front mounted connectors 646a-646j for the quadrant 1 and 2 forwarding cards.

Both mid-planes 622a and 622b include back mounted connectors 648a-648d for the switch fabric cards and back mounted connectors 650a-650d for the internal control cards. Mid-plane 622b further includes front, reverse mounted connectors 652a-652j for the quadrant 3 and 4 forwarding cards and back mounted connectors 654a-654p for the quadrant 3 and 4 universal port cards. In addition, mid-plane 622b also includes front, reverse mounted connectors 656a-656d for the quadrant 3 and 4 cross-connection cards and front mounted connectors 658a-658b for the auxiliary processor cards.

Combining both physical layer switch/router subsystems and upper layer switch/router subsystems in one network device allows for intelligent layer 1 switching. For example, the network device may be used to establish dynamic network connections on the layer 1 network to better utilize resources as service subscriptions change. In addition, network management is greatly simplified since the layer 1 and multiple upper layer networks may be managed by the same network management system and grooming fees are eliminated. Combining the physical layer switch/router and upper layer switch/routers into a network device that fits into one telco rack provides a less expensive network device and saves valuable telco site space.

Splitting the cross-connection function into four separate cards / quadrants enables the cross-connection routing requirements to be spread between the two mid-planes and alleviates the need to route cross-connection signals through the center of the device where the switch fabric is routed. In addition, segmenting the cross-connection function into multiple, independent subsystems allows customers / network managers to add functionality to network device 540 in pieces and in accordance with network service subscriptions. When a network device is first installed, a network manager may need only a few port cards and forwarding cards to service network customers. The modularity of network device 540 allows the network manager to purchase and install only one cross-connection card and the required number of port and forwarding cards. As the network becomes more subscribed, the network manager may add forwarding cards and port cards and eventually additional cross-connection cards. Since network devices are often very expensive, this modularity allows network managers to spread the cost of the system out in accordance with new service requests. The fees paid by customers to the network manager for the new services can then be applied to the cost of the new cards.

Although the embodiment describes the use of two mid-planes, it should be understood that more than two mid-planes may be used. Similarly, although the embodiment described flipped / reversed the forwarding cards and cross-connection cards in the lower half of the chassis, alternatively, the forwarding cards and cross-connection cards in the upper half of the chassis could be flipped.

Distributed Switch Fabric:

A network device having a distributed switch fabric locates a portion of the switch fabric functionality on cards separate from the remaining / central switch fabric functionality. For example, a portion of the switch fabric may be distributed on each forwarding card. There are a number of difficulties associated with distributing a portion of the switch fabric. For instance, distributing the switch fabric makes mid-plane / back-plane routing more difficult which further increases the difficulty of fitting the network device into one telco rack, switch fabric redundancy and timing are also made more difficult, valuable forwarding card space must be allocated for switch fabric components and the cost of each forwarding card is increased. However, since the entire switch fabric need not be included in a minimally configured network device, the cost of the minimal configuration is reduced allowing network service providers to more quickly recover the initial cost of the device. As new services are requested, additional functionality, including both forwarding cards (with additional switch fabric functionality) and universal port cards may be added to the network device to handle the new requests, and the fees for the new services may be applied to the cost of the additional functionality. Consequently, the cost of the network device more closely tracks the service fees received by network providers.

Referring again to Fig. 36, as described above, each forwarding card (e.g., 546c) includes traffic management chips (e.g., 588a-588n and 590a-590b) that ensure high priority network data / traffic (e.g., voice) is transferred faster than lower priority traffic (e.g., e-mail). Each forwarding card also includes switch fabric interface (SFIF) chips (e.g., 589a-589n) that transfer network data between the traffic management chips and the switch fabric cards 570a-570b.

Referring also to Fig. 43, forwarding card 546c includes traffic management (TM) chips 588n and 590a and SFIF chips 589, and forwarding card 550a includes traffic management chips 659a and 659b and SFIF chips 660. (Fig. 43 includes only two forwarding cards for convenience but it is to be understood that many forwarding cards may be included in a network device as shown in Fig. 35.) SFIF chips 589 and 660 on both boards include a switch fabric interface (SIF) chip 661, data slice chips 662a-662f, an enhanced port processor (EPP) chip 664 and a local timing subsystem (LTS) 665. The SFIF chips receive data from ingress TM chips 588n and 659a and

forward it to the switch fabric cards 570a – 570b (Fig. 36). Similarly, the SFIF chips receive data from the switch fabric cards and forward it to the egress TM chips 590a and 659b.

Due to the size and complexity of the switch fabric, each switch fabric card 570a-570b may include multiple separate cards. In one embodiment, each switch fabric card 570a-570b includes a control card 666 and four data cards 668a-668d. A scheduler chip 670 on control card 666 works with the EPP chips on each of the forwarding cards to transfer network data between the data slice chips on the forwarding cards through cross-bar chips 672a-672l (only chips 672a-672f are shown) on data cards 668a-668d. Each of the data slice chips on each of the forwarding cards is connected to two of the cross-bar chips on the data cards. Switch fabric control card 666 and each of the switch fabric data cards 668a-668d also include a switch fabric local timing subsystem (LTS) 665, and a switch fabric central timing subsystem (CTS) 673 on control card 666 provides a start of segment (SOS) reference signal to each LTS 665 on each of the forwarding cards and switch fabric cards.

The traffic management chips perform upper level network traffic management within the network device while scheduler chip 670 on control card 666 performs the lower level data transfer between forwarding cards. The traffic management chips determine the priority of received network data and then forward the highest priority data to SIF chips 661. The traffic management chips include large buffers to store lower priority data until higher priority data has been transferred. The traffic management chips also store data in these buffers when the local EPP chip indicates that data transfers are to be stopped (i.e., back pressure). The scheduler chip works with the EPP chips to stop or hold-off data transfers when necessary, for example, when buffers on one forwarding card are close to full, the local EPP chip sends notice to each of the other EPP chips and the scheduler to hold off sending more data. Back pressure may be applied to all forwarding cards when a new switch fabric control card is added to the network device, as described below.

The traffic management chips forward network data in predefined segments to the SIF chips. In the case of ATM data, each ATM cell is a segment. In the case of IP and MPLS, where the amount of network data in each packet may vary, the data is first

arranged into appropriately sized segments before being sent to the SIF chips. This may be accomplished through segmentation and reassembly (SAR) chips (not shown).

When the SIF chip receives a segment of network data, it organizes the data into a segment consistent with that expected by the switch fabric components, including any required header information. The SIF chip may be a PMC9324-TC chip available from Extreme Packet Devices (EPD), a subsidiary of PMC-Sierra, and the data slice chips may be PM9313-HC chips and the EPP chip may be a PM9315-HC chip available from Abrizio, also a subsidiary of PMC-Sierra. In this case, the SIF chip organizes each segment of data – including header information -- in accordance with a line-card-to-switch two (LCS-2) protocol. The SIF chip then divides each data segment into twelve slices and sends two slices to each data slice chip 662a-662f. Two slices are sent because each data slice chip includes the functionality of two data slices.

When the data slice chips receive the LCS segments, the data slice chips strip off the header information, including both a destination address and quality of service (QoS) information, and send the header information to the local EPP chip. Alternatively, the SIF chip may send the header information directly to the EPP chip and send only data to the data slice chips. However, the manufacturer teaches that the SIF chip should be on the forwarding card and the EPP and data slice chips should be on a separate switch fabric card within the network device or in a separate box connected to the network device. Minimizing connections between cards is important, and where the EPP and data slice chips are not on the same card as the SIF chips, the header information is sent with the data by the SIF chip to reduce the required inter-card connections, and the data slice chips then strip off this information and send it to the EPP chip.

The EPP chips on all of the forwarding cards communicate and synchronize through cross-bar chips 674a-674b on control card 666. For each time interval (e.g., every 40 nanoseconds, "ns"), the EPP chips inform the scheduler chip as to which data segment they would like to send and the data slice chips send a segment of data previously set up by the scheduler and EPP chips. The EPP chips and the scheduler use the destination addresses to determine if there are any conflicts, for example, to determine

if two or more forwarding cards are trying to send data to the same forwarding card. If a conflict is found, then the quality of service information is used to determine which forwarding card is trying to send the higher priority data. The highest priority data will likely be sent first. However, the scheduler chips include an algorithm that takes into account both the quality of service and a need to keep the switch fabric data cards 668a-668d full (maximum data through put). Where a conflict exists, the scheduler chip may inform the EPP chip to send a different, for example, lower priority, data segment from the data slice chip buffers or to send an empty data segment during the time interval.

Scheduler chip 670 informs each of the EPP chips which data segment is to be sent and received in each time interval. The EPP chips then inform their local data slice chips as to which data segments are to be sent in each interval and which data segments will be received in each interval. As previously mentioned, the forwarding cards each send and receive data. The data slice chips include small buffers to hold certain data (e.g., lower priority) while other data (e.g., higher priority) data is sent and small buffers to store received data. The data slice chips also include header information with each segment of data sent to the switch fabric cards. The header information is used by cross-bar chips 672a-672l (only cross-bar chips 672a-672f are shown) to switch the data to the correct forwarding card. The cross-bar chips may be PM9312-UC chips and the scheduler chip may be a PM9311-UC chip both of which are available from Abrizio.

Specifications for the EPD, Abrizio and PMC-Sierra chips may be found at www.pmc-sierra.com and are hereby incorporated herein by reference.

Distributed Switch Fabric Timing:

As previously mentioned, a segment of data (e.g., an ATM cell) is transferred between the data slice chips through the cross-bar chips every predetermined time interval. In one embodiment, this time interval is 40ns and is established by a 25MHz start of segment (SOS) signal. A higher frequency clock (e.g., 200 MHz, having a 5ns time interval) is used by the data slice and cross-bar chips to transfer the bits of data within each segment such that all the bits of data in a segment are transferred within one 40ns interval. More specifically, in one embodiment, each switch fabric

component multiplies the 200 MHz clock signal by four to provide an 800 MHz internal clock signal allowing data to be transferred through the data slice and cross-bar components at 320 Gbps. As a result, every 40ns one segment of data (e.g., an ATM cell) is transferred. It is crucial that the EPP, scheduler, data slice and cross-bar chips transfer data according to the same / synchronized timing signals (e.g., clock and SOS), including both frequency and phase. Transferring data at different times, even slightly different times, may lead to data corruption, the wrong data being sent and/or a network device crash.

When distributed signals (e.g., reference SOS or clock signals) are used to synchronize actions across multiple components (e.g., the transmission of data through a switch fabric), any time-difference in events (e.g., clock pulse) on the distributed signals is generally termed "skew". Skew between distributed signals may result in the actions not occurring at the same time, and in the case of transmission of data through a switch fabric, skew can cause data corruption and other errors. Many variables can introduce skew into these signals. For example, components used to distribute the clock signal introduce skew, and etches on the mid-plane(s) introduce skew in proportion to the differences in their length (e.g., about 180 picoseconds per inch of etch in FR 4 printed circuit board material).

To minimize skew, one manufacturer teaches that all switch fabric components (i.e., scheduler, EPP, data slice and cross-bar chips) should be located on centralized switch fabric cards. That manufacturer also suggests distributing a central clock reference signal (e.g., 200 MHz) and a separate SOS signal (e.g., 25 MHz) to the switch fabric components on the switch fabric cards. Such a timing distribution scheme is difficult but possible where all the components are on one switch fabric card or on a limited number of switch fabric cards that are located near each other within the network device or in a separate box connected to the network device. Locating the boards near each other within the network device or in a separate box allows etch lengths on the mid-plane for the reference timing signals to be more easily matched and, thus, introduce less skew.

When the switch fabric components are distributed, maintaining a very tight skew becomes difficult due to the long lengths of etches required to reach some of the

distributed cards and the routing difficulties that arise in trying to match the lengths of all the etches across the mid-plane(s). Because the clock signal needs to be distributed not only to the five switch fabric cards but also the forwarding cards (e.g., twenty), it becomes a significant routing problem to distribute all clocks to all loads with a fixed etch length.

Since timing is so critical to network device operation, typical network devices include redundant central timing subsystems. Certainly, the additional reference timing signals from a redundant central timing subsystem to each of the forwarding cards and switch fabric cards create further routing difficulties. In addition, if the two central timing subsystems (i.e., sources) are not synchronous with matched distribution etches, then all of the loads (i.e., LTSs) must use the same reference clock source to avoid introducing clock skew – that is, unless both sources are synchronous and have matched distribution networks, the reference timing signals from both sources are likely to be skewed with respect to each other and, thus, all loads must use the same source / reference timing signal or be skewed with respect to each other.

A redundant, distributed switch fabric greatly increases the number of reference timing signals that must be routed over the mid-planes and yet remain accurately synchronized. In addition, since the timing signals must be sent to each card having a distributed switch fabric, the distance between the cards may vary greatly and, thus, make matching the lengths of timing signal etches on the mid-planes difficult. Further, the lengths of the etches for the reference timing signals from both the primary and redundant central timing subsystems must be matched. Compounding this with a fast clock signal and low skew component requirements makes distributing the timing very difficult.

The network device of the present invention, though difficult, includes two synchronized central timing subsystems (CTS) 673 (one is shown in Fig. 43). The etch lengths of reference timing signals from both central timing subsystems are matched to within, for example, +/- 50 mils, and both central timing subsystems distribute only reference start of segment (SOS) signals to a local timing subsystem (LTS) 665 on each forwarding card and switch fabric card. The LTSs use the SOS reference signals to generate both an SOS signal and a higher frequency clock signal.

This adds components and complexity to the LTSs, however, distributing only the SOS reference signals and not both the SOS and clock reference signals significantly reduces the number of reference timing signals that must be routed across the mid-plane on matched etch lengths.

Both electro-magnetic radiation and electro-physical limitations prevent the 200 MHz reference clock signal from being widely distributed as required in a network device implementing distributed switch fabric subsystems. Such a fast reference clock increases the overall noise level generated by the network device and wide distribution may cause the network device to exceed Electro-Magnetic Interference (EMI) limitations. Clock errors are often measured as a percentage of the clock period, the smaller the clock period (5ns for a 200 MHz clock), the larger the percentage of error a small skew can cause. For example, a skew of 3ns represents a 60% error for a 5ns clock period but only a 7.5% error for a 40ns clock period. Higher frequency clock signals (e.g., 200 MHz) are susceptible to noise error and clock skew. The SOS signal has a larger clock period than the reference clock signal (40ns versus 5ns) and, thus, is less susceptible to noise error and reduces the percentage of error resulting from clock skew.

As previously mentioned, the network device may include redundant switch fabric cards 570a and 570b (Fig. 36) and as described above with reference to Fig. 43, each switch fabric card 570a and 570b may include a control card and four or more data cards. Referring to Fig. 44, network device 540 may include switch fabric control card 666 (part of central switch fabric 570a) and redundant switch fabric control card 667 (part of redundant switch fabric 570b). Each control card 666 and 667 includes a central timing subsystem (CTS) 673. One CTS behaves as the master and the other CTS behaves as a slave and locks its output SOS signal to the master's output SOS signal. In one embodiment, upon power-up or system re-boot the CTS on the primary switch fabric control card 666 begins as the master and if a problem occurs with the CTS on the primary control card, then the CTS on redundant control card 667 takes over as master without requiring a switch over of the primary switch fabric control card.

Still referring to Fig. 44, each CTS sends a reference SOS signal to the LTSs on each forwarding card, switch fabric data cards 668a-668d and redundant switch fabric data cards 669a-669b. In addition, each CTS sends a reference SOS signal to the LTS on its own switch fabric control card and the LTS on the other switch fabric control card. As described in more detail below, each LTS then selects which reference SOS signal to use. Each CTS 673 also sends a reference SOS signal to the CTS on the other control card. The master CTS ignores the reference SOS signal from the slave CTS but the slave CTS locks its reference SOS signal to the reference SOS signal from the master, as described below. Locking the slave SOS signal to the master SOS signal synchronizes the slave signal to the master signal such that in the event that the master CTS fails and the LTSs switchover to the slave CTS reference SOS signal and the slave CTS becomes the master CTS, minimal phase change and no signal disruption is encountered between the master and slave reference SOS signals received by the LTSs.

Each of the CTS reference SOS signals sent to the LTSs and the other CTS over mid-plane etches are the same length (i.e., matched) to avoid introducing skew. The CTS may be on its own independent card or any other card in the system. Even when it is located on a switch fabric card, such as the control card, that has an LTS, the reference SOS signal is routed through the mid-plane with the same length etch as the other reference SOS signals to avoid adding skew.

Central Timing Subsystem (CTS):

Referring to Fig. 45, central timing subsystem (CTS) 673 includes a voltage controlled crystal oscillator (VCXO) 676 that generates a 25MHz reference SOS signal 678. The SOS signal must be distributed to each of the local timing subsystems (LTSs) and is, thus, sent to a first level clock driver 680 and then to second level clock drivers 682a-682d that output reference SOS signals SFC_BENCH_FB and SFC_REF1 – SFC_REFn. SFC_BENCH_FB is a local feedback signal returned to the input of the CTS. One of SFC_REF1 - SFC_REFn is sent to each LTS, the other CTS, which receives it on SFC_SYNC, and one is routed over a mid-plane and returned as a feedback signal SFC_FB to the input of the CTS that generated it. Additional levels of clock drivers may be added as the number of necessary reference SOS signals increases.

VCXO 676 may be a VF596ES50 25MHz LVPECL available from Conner-Winfield. Positive Emitter Coupled Logic (PECL) is preferred over Transistor-Transistor Logic (TTL) for its lower skew properties. In addition, though it requires two etches to transfer a single clock reference -- significantly increasing routing resources --, differential PECL is preferred over PECL for its lower skew properties and high noise immunity. The clock drivers are also differential PECL and may be one to ten (1:10) MC100 LVEP111 clock drivers available from On Semiconductor. A test header 681 may be connected to clock driver 680 to allow a test clock to be input into the system.

Hardware control logic 684 determines (as described below) whether the CTS is the master or slave, and hardware control logic 684 is connected to a multiplexor (MUX) 686 to select between a predetermined voltage input (i.e., master voltage input) 688a and a slave VCXO voltage input 688b. When the CTS is the master, hardware control logic 684 selects predetermined voltage input 688a from discrete bias circuit 690 and slave VCXO voltage input 688b is ignored. The predetermined voltage input causes VCXO 676 to generate a constant 25MHz SOS signal; that is, the VCXO operates as a simple oscillator.

Hardware control logic may be implemented in a field programmable gate array (FPGA) or a programmable logic device (PLD). MUX 686 may be a 74CBTLV3257 FET 2:1 MUX available from Texas Instruments.

When the CTS is the slave, hardware control logic 684 selects slave VCXO voltage signal 688b. This provides a variable voltage level to the VCXO that causes the output of the VCXO to track or follow the SOS reference signal from the master CTS. Referring still to Fig. 45, the CTS receives the SOS reference signal from the other CTS on SFC_SYNC. Since this is a differential PECL signal, it is first passed through a differential PECL to TTL translator 692 before being sent to MUX 697a within dual MUX 694. In addition, two feedback signals from the CTS itself are supplied as inputs to the CTS. The first feedback signal SFC_FB is an output signal (e.g., one of SFC_REF1-SFC_REFn) from the CTS itself which has been sent out to the mid-plane and routed back to the switch fabric control card. This is done so that the feedback signal used by the CTS experiences identical conditions as the reference

SOS signal delivered to the LTSs and skew is minimized. The second feedback signal SFC_BENCH_FB is a local signal from the output of the CTS, for example, clock driver 682a. SFC_BENCH_FB may be used as the feedback signal in a test mode, for example, when the control card is not plugged into the network device chassis and SFC_SB is unavailable. SFC_BENCH_FB and SFC_FB are also differential PECL signals and must be sent through translators 693 and 692, respectively, prior to being sent to MUX 697b within dual MUX 694. Hardware control logic 684 selects which inputs are used by MUX 694 by asserting signals on REF_SEL(1:0) and FB_SEL(1:0). In regular use, inputs 696a and 696b from translator 692 are selected. In test modes, grounded inputs 695a, test headers 695b or local feedback signal 698 from translator 693 may be selected. Also in regular use (and in test modes where a clock signal is not inserted through the test headers), copies of the selected input signals are provided on the test headers.

The reference output 700a and the feedback output 700b are then sent from the MUX to phase detector circuit 702. The phase detector compares the rising edge of the two input signals to determine the magnitude of any phase shift between the two. The phase detector then generates variable voltage pulses on outputs 704a and 704b representing the magnitude of the phase shift. The phase detector outputs are used by discrete logic circuit 706 to generate a voltage on a slave VCXO voltage signal 688b representing the magnitude of the phase shift. The voltage is used to speed up or slow down (i.e., change the phase of) the VCXO's output SOS signal to allow the output SOS signal to track any phase change in the reference SOS signal from the other CTS (i.e., SFC_SYNC). The discrete logic components implement filters that determine how quickly or slowly the VCXO's output will track the change in phase detected on the reference signal. The combination of the dual MUX, phase detector, discrete logic, VCXO, clock drivers and feedback signal forms a phase locked loop (PLL) circuit allowing the slave CTS to synchronize its reference SOS signal to the master CTS reference SOS signal. MUX 686 and discrete bias circuit 690 are not found in phase locked loop circuits.

The phase detector circuit may be implemented in a programmable logic device (PLD), for example a MACH4LV-32 available from Lattice/Vantis Semiconductor. Dual MUX 694 may be implemented in the same PLD. Preferably, however, dual

MUX 694 is an SN74CBTLV3253 available from Texas Instruments, which has better skew properties than the PLD. The differential PECL to TTL translators may be MC100EPT23 dual differential PECL/TTL translators available from On Semiconductor.

Since quick, large phase shifts in the reference signal are likely to be the results of failures, the discrete logic implements a filter, and for any detected phase shift, only small incremental changes over time are made to the voltage provided on slave VCXO control signal 688b. As one example, if the reference signal from the master CTS dies, the slave VCXO control signal 688b only changes phase slowly over time meaning that the VCXO will continue to provide a reference SOS signal. If the reference signal from the master CTS is suddenly returned, the slave VCXO control signal 688b again only changes phase slowly over time to cause the VCXO signal to re-synchronize with the reference signal from the master CTS. This is a significant improvement over distributing a clock signal directly to components that use the signal because, in the case of direct clock distribution, if one clock signal dies (e.g., broken wire), then the components connected to that signal stop functioning causing the entire switch fabric to fail.

Slow phase changes on the reference SOS signals from both the master and slave CTSs are also important when LTSs switch over from using the master CTS reference signal to using the slave CTS reference signal. For example, if the reference SOS signal from the master CTS dies or other problems are detected (e.g., a clock driver dies), then the slave CTS switches over to become the master CTS and each of the LTSs begin using the slave CTS' reference SOS signal. For these reasons, it is important that the slave CTS reference SOS signal be synchronized to the master reference signal but not quickly follow large phase shifts in the master reference signal.

It is not necessary for every LTS to use the reference SOS signals from the same CTS. In fact, some LTSs may use reference SOS signals from the master CTS while one or more are using the reference SOS signals from the slave CTS. In general, this is a transitional state prior to or during switch over. For example, one or more LTSs may

start using the slave CTS's reference SOS signal prior to the slave CTS switching over to become the master CTS.

It is important for both the CTSs and the LTSs to monitor the activity of the reference SOS signals from both CTSs such that if there is a problem with one, the LTSs can begin using the other SOS signal immediately and/or the slave CTS can quickly become master. Reference output signal 700a – the translated reference SOS signal sent from the other CTS and received on SFC_SYNC -- is sent to an activity detector circuit 708. The activity detector circuit determines whether the signal is active – that is, whether the signal is "stuck at" logic 1 or logic 0. If the signal is not active (i.e., stuck at logic 1 or 0), the activity detector sends a signal 683a to hardware control logic 684 indicating that the signal died. The hardware control logic may immediately select input 688a to MUX 686 to change the CTS from slave to master. The hardware control logic also sends an interrupt to a local processor 710 and software being executed by the processor detects the interrupt. Hardware control allows the CTS switch over to happen very quickly before a bad clock signal can disrupt the system.

Similarly, an activity detector 709 monitors the output of the first level clock driver 680 regardless of whether the CTS is master or slave. Instead, the output of one the second level clock drivers could be monitored, however, a failure of a different second level clock will not be detected. SFC_REF_ACTIVITY is sent from the first level clock driver to differential PECL to TTL translator 693 and then as FABRIC_REF_ACTIVITY to activity detector 709. If activity detector 709 determines that the signal is not active, which may indicate that the clock driver, oscillator or other component(s) within the CTS have failed, then it sends a signal 683b to the hardware control logic. The hardware control logic asserts KILL_CLKTREE to stop the clock drivers from sending any signals and notifies a processor chip 710 on the switch fabric control card through an interrupt. Software being executed by the processor chip detects the interrupt. The slave CTS activity detector 708 detects a dead signal from the master CTS either before or after the hardware control logic sends KILL_CLKTREE and asserts error signal 683a to cause the hardware control logic to change the input selection on MUX 686 from 688b to 688a to become the master CTS. As described below, the LTSs also detect a dead

signal from the master CTS either before or after the hardware control logic sends KILL_CLKTREE and switch over to the reference SOS signal from the slave CTS either before or after the slave CTS switches over to become the master.

As previously mentioned, in the past, a separate, common clock selection signal or etch was sent to each card in the network device to indicate whether to use the master or slave clock reference signal. This approach required significant routing resources, was under software control and resulted in every load selecting the same source at any given time. Hence, if a clock signal problem was detected, components had to wait for the software to change the separate clock selection signal before beginning to use the standby clock signal and all components (i.e., loads) were always locked to the same source. This delay can cause data corruption errors, switch fabric failure and a network device crash.

Forcing a constant logic one or zero (i.e., "killing") clock signals from a failed source and having hardware in each LTS and CTS detect inactive (i.e., "dead" or stuck at logic one or zero) signals allows the hardware to quickly begin using the standby clock without the need for software intervention. In addition, if only one clock driver (e.g., 682b) dies in the master CTS, LTSs receiving output signals from that clock driver may immediately begin using signals from the slave CTS clock driver while the other LTSs continue to use the master CTS. Interrupts to the processor from each of the LTSs connected to the failed master CTS clock driver allow software, specifically the SRM, to detect the failure and initiate a switch over of the slave CTS to the master CTS. The software may also override the hardware control and force the LTSs to use the slave or master reference SOS signal.

When the slave CTS switches over to become the master CTS, the remaining switch fabric control card functionality (e.g., scheduler and cross-bar components) continue operating. The SRM (described above) decides – based on a failure policy -- whether to switch over from the primary switch fabric control card to the secondary switch fabric control card. There may be instances where the CTS on the secondary switch fabric control card operates as the master CTS for a period of time before the network device switches over from the primary to the secondary switch fabric control card, or instead, there may be instances where the CTS on the secondary switch fabric control

card operates as the master CTS for a period of time and then the software directs the hardware control logic on both switch fabric control cards to switch back such that the CTS on the primary switch fabric control card is again master. Many variations are possible since the CTS is independent of the remaining functionality on the switch fabric control card.

Phase detector 702 also includes an out of lock detector that determines whether the magnitude of change between the reference signal and the feedback signal is larger than a predetermined threshold. When the CTS is the slave, this circuit detects errors that may not be detected by activity detector 708 such as where the reference SOS signal from the master CTS is failing but is not dead. If the magnitude of the phase change exceeds the predetermined threshold, then the phase detector asserts an OOL signal to the hardware control logic. The hardware control logic may immediately change the input to MUX 686 to cause the slave CTS to switch over to Master CTS and send an interrupt to the processor, or the hardware control logic may only send the interrupt and wait for software (e.g., the SRM) to determine whether the slave CTS should switch over to master.

Master / Slave CTS Control:

In order to determine which CTS is the master and which is the slave, hardware control logic 684 implements a state machine. Each hardware control logic 684 sends an IM_THE_MASTER signal to the other hardware control logic 684 which is received as a YOU_THE_MASTER signal. If the IM_THE_MASTER signal – and, hence, the received YOU_THE_MASTER signal -- is asserted then the CTS sending the signal is the master (and selects input 688a to MUX 686, Fig. 45) and the CTS receiving the signal is the slave (and selects input 688b to MUX 686). Each IM_THE_MASTER / YOU_THE_MASTER etch is pulled down to ground on the mid-planes such that if one of the CTSs is missing, the YOU_THE_MASTER signal received by the other CTS will be a logic 0 causing the receiving CTS to become the master. This situation may arise, for example, if a redundant control card including the CTS is not inserted within the network device. In addition, each of the hardware control logics receive SLOT_ID signals from pull-down/pull-up resistors on the chassis mid-plane indicating the slot in which the switch fabric control card is inserted.

Referring to Fig. 46, on power-up or after a system or card or CTS re-boot, the hardware control logic state machine begins in INIT/RESET state 0 and does not assert IM_THE_MASTER. If the SLOT_ID signals indicate that the control card is inserted in a preferred slot (e.g., slot one), and the received YOU_THE_MASTER is not asserted (i.e., 0), then the state machine transitions to the ONLINE state 3 and the hardware control logic asserts IM_THE_MASTER indicating its master status to the other CTS and selects input 688a to MUX 686. While in the ONLINE state 3, if a failure is detected or the software tells the hardware logic to switch over, the state machine enters the OFFLINE state 1 and the hardware control logic stops asserting IM_THE_MASTER and asserts KILL_CLKTREE. While in the OFFLINE state 1, the software may reset or re-boot the control card or just the CTS and force the state machine to enter the STANDBY state 2 as the slave CTS and the hardware control logic stops asserting KILL_CLKTREE and selects input 688b to MUX 686.

While in INIT/RESET state 0, if the SLOT_ID signals indicate that the control card is inserted in a non-preferred slot, (e.g., slot 0), then the state machine will enter STANDBY state 2 as the slave CTS and the hardware control logic will not assert IM_THE_MASTER and will select input 688b to MUX 686. While in INIT/RESET state 0, even if the SLOT_ID signals indicate that the control card is inserted in the preferred slot, if YOU_THE_MASTER is asserted, indicating that the other CTS is master, then the state machine transfers to STANDBY state 2. This situation may arise after a failure and recovery of the CTS in the preferred slot (e.g., reboot, reset or new control card).

While in the STANDBY state 2, if the YOU_THE_MASTER signal becomes zero (i.e., not asserted), indicating that the master CTS is no longer master, the state machine will transition to ONLINE state 3 and the hardware control logic will assert IM_THE_MASTER and select input 688a to MUX 686 to become master. While in ONLINE state 3, if the YOU_THE_MASTER signal is asserted and SLOT_ID indicating slot 0 the state machine enters STANDBY state 2 and the hardware control logic stops asserting IM_THE_MASTER and selects input 688b to MUX 686. This is the situation where the original master CTS is back up and running. The software

may reset the state machine at any time or set the state machine to a particular state at any time.

Local Timing Subsystem:

Referring to Fig. 47, each local timing subsystem (LTS) 665 receives a reference SOS signal from each CTS on SFC_REFA and SFC_REFB. Since these are differential PECL signals, each is passed through a differential PECL to TTL translator 714a or 714b, respectively. A feedback signal SFC_FB is also passed from the LTS output to both translators 714a and 714b. The reference signal outputs 716a and 716b are fed into a first MUX 717 within dual MUX 718, and the feedback signal outputs 719a and 719b are fed into a second MUX 720 within dual MUX 718. LTS hardware control logic 712 controls selector inputs REF_SEL (1:0) and FB_SEL (1:0) to dual MUX 718. With regard to the feedback signals, the LTS hardware control logic selects the feedback signal that went through the same translator as the reference signal that is selected to minimize the effects of any skew introduced by the two translators.

A phase detector 722 receives the feedback (FB) and reference (REF) signals from the dual MUX and, as explained above, generates an output in accordance with the magnitude of any phase shift detected between the two signals. Discrete logic circuit 724 is used to filter the output of the phase detector, in a manner similar to discrete logic 706 in the CTS, and provide a signal to VCXO 726 representing a smaller change in phase than that output from the phase detector. Within the LTSs, the VCXO is a 200 MHz oscillator as opposed to the 25MHz oscillator used in the CTS. The output of the VCXO is the reference switch fabric clock. It is sent to clock driver 728, which fans the signal out to each of the local switch fabric components. For example, on the forwarding cards, the LTSs supply the 200MHz reference clock signal to the EPP and data slice chips, and on the switch fabric data cards, the LTSs supply the 200 MHz reference clock signal to the cross-bar chips. On the switch fabric control card, the LTSs supply the 200 MHz clock signal to the scheduler and cross-bar components.

The 200 MHz reference clock signal from the VCXO is also sent to a divider circuit or component 730 that divides the clock by eight to produce a 25MHz reference SOS signal 731. This signal is sent to clock driver 732, which fans the signal out to each of

the same local switch fabric components that the 200 MHz reference clock signal was sent to. In addition, reference SOS signal 731 is provided as feedback signal SFC_FB to translator 714b. The combination of the dual MUX, phase detector, discrete logic, VCXO, clock drivers and feedback signal forms a phase locked loop circuit allowing the 200 MHz and 25MHz signals generated by the LTS to be synchronized to either of the reference SOS signals sent from the CTSs.

The divider component may be a SY100EL34L divider by Synergy Semiconductor Corporation.

Reference signals 716a and 716b from translator 714a are also sent to activity detectors 734a and 734b, respectively. These activity detectors perform the same function as the activity detectors in the CTSs and assert error signals ref_a_los or ref_b_los to the LTS hardware control logic if reference signal 716a or 716b, respectively, die. On power-up, reset or reboot, a state machine (Fig. 48) within the LTS hardware control logic starts in INIT/RESET state 0. Arbitrarily, reference signal 716a is the first signal considered. If activity detector 734a is not sending an error signal (i.e., ref_a_los is 0), indicating that that reference signal 716a is active, then the state machine changes to REF_A state 2 and sends signals over REF_SEL(1:0) to MUX 717 to select reference input 716a and sends signals over FB_SEL(1:0) to MUX 720 to select feedback input 719a. While in INIT/RESET state 0, if ref_a_los is asserted, indicating no signal on reference 716a, and if ref_b_los is not asserted, indicating there is a signal on reference 716b, then the state machine changes to REF_B state 1 and changes REF_SEL(1:0) and FB_SEL(1:0) to select reference input 716b and feedback signal 719b.

While in REF_A state 2, if activity detector 734a detects a loss of reference signal 716a and asserts ref_a_los, the state machine will change to REF_B state 1 and change REF_SEL(1:0) and FB_SEL(1:0) to select inputs 716b and 719b. Similarly, while in REF_B state 1, if activity detector 734b detects a loss of signal 716b and asserts ref_b_los, the state machine will change to REF_A state 2 and change REF_SEL(1:0) and FB_SEL(1:0) to select inputs 716a and 719a. While in either REF_A state 2 or REF_B state 1, if both ref_a_los and ref_b_los are asserted, indicating that both reference SOS signals have died, the state machine changes back

to INIT/RESET state 0 and change REF_SEL(1:0) and FB_SEL(1:0) to select no inputs or test inputs 736a and 736b or ground 738. For a period of time, the LTS will continue to supply a clock and SOS signal to the switch fabric components even though it is receiving no input reference signal.

When ref_a_los and/or ref_b_los are asserted, the LTS hardware control logic notifies its local processor 740 through an interrupt. The SRM will decide, based on a failure policy, what actions to take, including whether to switch over from the master to slave CTS. Just as the phase detector in the CTS sends an out of lock signal to the CTS hardware control logic, the phase detector 722 also sends an out of lock signal OOL to the LTS hardware control logic if the magnitude of the phase difference between the reference and feedback signals exceeds a predetermined threshold. If the LTS hardware receives an asserted OOL signal, it notifies its local processor (e.g., 740) through an interrupt. The SRM will decide based on a failure policy what actions to take.

Shared LTS Hardware:

In the embodiment described above, the switch fabric data cards are four independent cards. More data cards may also be used. Alternatively, all of the cross-bar components may be located on one card. As another alternative, half of the cross-bar components may be located on two separate cards and yet attached to the same network device faceplate and share certain components. A network device faceplate is something the network manager can unlatch and pull on to remove cards from the network device. Attaching two switch fabric data cards to the same faceplate effectively makes them one board since they are added to and removed from the network device together. Since they are effectively one board, they may share certain hardware as if all components were on one physical card. In one embodiment, they may share a processor, hardware control logic and activity detectors. This means that these components will be on one of the physical cards but not on the other and signals connected to the two cards allow activity detectors on the one card to monitor the reference and feedback signals on the other card and allow the hardware control logic on the one card to select the inputs for dual MUX 718 on the other card.

Scheduler:

Another difficulty with distributing a portion of the switch fabric functionality involves the scheduler component on the switch fabric control cards. In current systems, the entire switch fabric, including all EPP chips, are always present in a network device. Registers in the scheduler component are configured on power-up or re-boot to indicate how many EPP chips are present in the current network device, and in one embodiment, the scheduler component detects an error and switches over to the redundant switch fabric control card when one of those EPP chips is no longer active. When the EPP chips are distributed to different cards (e.g., forwarding cards) within the network device, an EPP chip may be removed from a running network device when the printed circuit board on which it is located is removed ("hot swap", "hot removal") from the network device. To prevent the scheduler chip from detecting the missing EPP chip as an error (e.g., a CRC error) and switching over to the redundant switch fabric control card, prior to the board being removed from the network device, software running on the switch fabric control card re-configures the scheduler chip to disable the scheduler chip's links to the EPP chip that is being removed.

To accomplish this, a latch 547 (Fig. 40) on the faceplate of each of the printed circuit boards on which a distributed switch fabric is located is connected to a circuit 742 (Fig. 44) also on the printed circuit board that detects when the latch is released. When the latch is released, indicating that the board is going to be removed from the network device, circuit 742 sends a signal to a circuit 743 on both switch fabric control cards indicating that the forwarding card is about to be removed. Circuit 743 sends an interrupt to the local processor (e.g., 710, Fig. 45) on the switch fabric control card. Software (e.g., slave SRM) being executed by the local processor detects the interrupt and sends a notice to software (e.g., master SRM) being executed by the processor (e.g., 24, Fig. 1) on the network device centralized processor card (e.g., 12, Fig. 1, 542 or 543, Fig. 35). The master SRM sends a notice to the slave SRMs being executed by the processors on the switch fabric data cards and forwarding cards to indicate the removal of the forwarding card. The redundant forwarding card switches over to become a replacement for the failed primary forwarding card. The master SRM also sends a notice to the slave SRM on the cross-connection card (e.g., 562-562b, 564a-564b, 566a-566b, 568a-565b, Fig. 35) to re-configure the connections between the port cards (e.g., 554a-554h, 556a-556h, 558a-

558h, 560a-560h, Fig. 35) and the redundant forwarding card. The slave SRM on the switch fabric control card re-configures the registers in the scheduler component to disable the scheduler's links to the EPP chip on the forwarding card that's being removed from the network device. As a result, when the forwarding card is removed, the scheduler will not detect an error due to a missing EPP chip.

Similarly, when a forwarding card is added to the network device, circuit 742 detects the closing of the latch and sends an interrupt to the processor. The slave SRM running on the local processor sends a notice to the Master SRM which then sends a notice to the slave SRMs being executed by the processors on the switch fabric control cards, data cards and forwarding cards indicating the presence of the new forwarding card. The slave SRM on the cross-connection cards may be re-configured, and the slave SRM on the switch fabric control card may re-configure the scheduler chip to establish links with the new EPP chip to allow data to be transferred to the newly added forwarding card.

Switch Fabric Control Card Switch-Over:

Typically, the primary and secondary scheduler components receive the same inputs, maintain the same state and generate the same outputs. The EPP chips are connected to both scheduler chips but only respond to the master / primary scheduler chip. If the primary scheduler or control card experiences a failure a switch over is initiated to allow the secondary scheduler to become the primary. When the failed switch fabric control card is re-booted, re-initialized or replaced, it and its scheduler component serve as the secondary switch fabric control card and scheduler component.

In currently available systems, a complex sequence of steps is required to "refresh" or synchronize the state of the newly added scheduler component to the primary scheduler component and for many of these steps, network data transfer through the switch fabric is temporarily stopped (i.e., back pressure). Stopping network data transfer may affect the availability of the network device. When the switch fabric is centralized and all on one board or only a few boards or in its own box, the refresh steps are quickly completed by one or only a few processors limiting the amount of time that network data is not transferred. When the switch fabric includes distributed switch fabric subsystems, the processors that are local to each of the distributed

switch fabric subsystems must take part in the series of steps. This may increase the amount of time that data transfer is stopped further affecting network device availability.

To limit the amount of time that data transfer is stopped in a network device including distributed switch fabric subsystems, the local processors each set up for a refresh while data is still being transferred. Communications between the processors take place over the Ethernet bus (e.g., 32, Fig. 1, 544, Fig. 35) to avoid interrupting network data transfer. When all processors have indicated (over the Ethernet bus) that they are ready for the refresh, the processor on the master switch fabric control card stops data transfer and sends a refresh command to each of the processors on the forwarding cards and switch fabric cards. Since all processors are waiting to complete the refresh, it is quickly completed. Each processor notifies the processor on the master switch fabric control card that the refresh is complete, and when all processors have completed the refresh, the master switch fabric control card re-starts the data transfer.

During the time in which the data transfer is stopped, the buffers in the traffic management chips are used to store data coming from external network devices. It is important that the data transfer be complete quickly to avoid overrunning the traffic management chip buffers.

Since the switch over of the switch fabric control cards is very complex and requires that data transfer be stopped, even if briefly, it is important that the CTSs on each switch fabric control card be independent of the switch fabric functionality. This independence allows the master CTS to switch over to the slave CTS quickly and without interrupting the switch fabric functionality or data transmission.

As described above, locating the EPP chips and data slice chips of the switch fabric subsystem on the forwarding cards is difficult and against the teachings of a manufacturer of these components. However, locating these components on the forwarding cards allows the base network device – that is, the minimal configuration – to include only a necessary portion of the switching fabric reducing the cost of a minimally configured network device. As additional forwarding cards are added to

the minimal configuration -- to track an increase in customer demand -- additional portions of the switch fabric are simultaneously added since a portion of the switch fabric is located on each forwarding card. Consequently, switch fabric growth tracks the growth in customer demands and fees. Also, typical network devices include 1:1 redundant switch fabric subsystems. However, as previously mentioned, the forwarding cards may be 1:N redundant and, thus, the distributed switch fabric on each forwarding card is also 1:N redundant further reducing the cost of a minimally configured network device.

External Network Data Transfer Timing:

In addition to internal switch fabric timing, a network device must also include external network data transfer timing to allow the network device to transfer network data synchronously with other network devices. Generally, multiple network devices in the same service provider site synchronize themselves to Building Integrated Timing Supply (BITS) lines provided by a network service provider. BITS lines are typically from highly accurate stratum two clock sources. In the United States, standard T1 BITS lines (2.048 MHz) are provided, and in Europe, standard E1 BITS lines (1.544 MHz) are provided. Typically, a network service provider provides two T1 lines or two E1 lines from different sources for redundancy. Alternatively, if there are no BITS lines or when network devices in different sites want to synchronously transfer data, one network device may extract a timing signal received on a port connected to the other network device and use that timing signal to synchronize its data transfers with the other network device.

Referring to Fig. 49, controller card 542b and redundant controller card 543b each include an external central timing subsystem (EX CTS) 750. Each EX CTS receives BITS lines 751 and provide BITS lines 752. In addition, each EX CTS receives a port timing signal 753 from each port card (554a-554h, 556a-556h, 558a-558h, 560a-560h, Fig. 35), and each EX CTS also receives an external timing reference signal 754 from itself and an external timing reference signal 755 from the other EX CTS.

One of the EX CTSs behaves as a master and the other EX CTS behaves as a slave. The master EX CTS may synchronize its output external reference timing signals to one of BITS lines 751 or one of the port timing signals 753, while the slave EX CTS

synchronizes its output external reference timing signals to the received master external reference timing signal 755. Upon a master EX CTS failure, the slave EX CTS may automatically switch over to become the master EX CTS or software may upon an error or at any time force the slave EX CTS to switch over to become the master EX CTS.

An external reference timing signal from each EX CTS is sent to each external local timing subsystem (EX LTS) 756 on cards throughout the network device, and each EX LTS generates local external timing signals synchronized to one of the received external reference timing signals. Generally, external reference timing signals are sent only to cards including external data transfer functionality, for example, cross connection cards 562a-562b, 564a-564b, 566a-566b and 568a-568b (Fig. 35) and universal port cards 554a-554h, 556a-556h, 558a-558h, 560a-560h.

In network devices having multiple processor components, an additional central processor timing subsystem is needed to generate processor timing reference signals to allow the multiple processors to synchronize certain processes and functions. The addition of both external reference timing signals (primary and secondary) and processor timing reference signals (primary and secondary) require significant routing resources. In one embodiment of the invention, the EX CTSs embed a processor timing reference signal within each external timing reference signal to reduce the number of timing reference signals needed to be routed across the mid-plane(s). The external reference timing signals are then sent to EX LTSs on each card in the network device having a processor component, for example, cross connection cards 562a-562b, 564a-564b, 566a-566b, 568a-568b, universal port cards 554a-554h, 556a-556h, 558a-558h, 560a-560h, forwarding cards 546a-546e, 548a-548e, 550a-550e, 552a-552e, switch fabric cards 666, 667, 668a-668d, 669a-669d (Fig. 44) and both the internal controller cards 542a, 543a (Fig. 41b) and external controller cards 542b and 543b.

All of the EX LTSs extract out the embedded processor reference timing signal and send it to their local processor component. Only the cross-connection cards and port cards use the external reference timing signal to synchronize external network data transfers. As a result, the EX LTSs include extra circuitry not necessary to the

function of cards not including external data transfer functionality, for example, forwarding cards, switch fabric cards and internal controller cards. The benefit of reducing the necessary routing resources, however, out weighs any disadvantage related to the excess circuitry. In addition, for the cards including external data transfer functionality, having one EX LTS that provides both local signals actually saves resources on those cards, and separate processor central timing subsystems are not necessary. Moreover, embedding the processor timing reference signal within the highly accurate, redundant external timing reference signal provides a highly accurate and redundant processor timing reference signal. Furthermore having a common EX LTS on each card allows access to the external timing signal for future modifications and having a common EX LTS, as opposed to different LTSs for each reference timing signal, results in less design time, less debug time, less risk, design re-use and simulation re-use.

Although the EX CTSs are described as being located on the external controllers 542b and 543b, similar to the switch fabric CTSs described above, the EX CTSs may be located on their own independent cards or on any other cards in the network device, for example, internal controllers 542a and 543a. In fact, one EX CTS could be located on an internal controller while the other is located on an external controller. Many variations are possible. In addition, just as the switch fabric CTSs may switch over from master to slave without affecting or requiring any other functionality on the local printed circuit board, the EX CTSs may also switch over from master to slave without affecting or requiring any other functionality on the local printed circuit board.

External Central Timing Subsystem (EX CTS):

Referring to Fig. 50, EX CTS 750 includes a T1/E1 framer/LIU 758 for receiving and terminating BITS signals 751 and for generating and sending BITS signals 752. Although T1/E1 framer is shown in two separate boxes in Fig. 50, it is for convenience only and may be the same circuit or component. In one embodiment, two 5431 T1/E1 Framer Line Interface Units (LIU) available from PMC-Sierra are used. The T1/E1 framer supplies 8KHz BITS_REF0 and BITS_REF1 signals and receives 8KHz BITS1_TXREF and BITS2_TXREF signals. A network administrator notifies NMS 60 (Fig. 35) as to whether the BITS signals are T1 or E1, and the NMS

notifies software running on the network device. Through signals 761 from a local processor, hardware control logic 760 within the EX CTS is configured for T1 or E1 and sends an T1E1_MODE signal to the T1/E1 framer indicating T1 or E1 mode. The T1/E1 framer then forwards BITS_REF0 and BITS_REF1 to dual MUXs 762a and 762b.

Port timing signals 753 are also sent to dual MUXs 762a and 762b. The network administrator also notifies the NMS as to which timing reference signals should be used, the BITS lines or the port timing signals. The NMS again notifies software running on the network device and through signals 761, the local processor configures the hardware control logic. The hardware control logic then uses select signals 764a and 764b to select the appropriate output signals from the dual MUXs.

Activity detectors 766a and 766b provide status signals 767a and 767b to the hardware control logic indicating whether the PRI_REF signal and the SEC_REF signal are active or inactive (i.e., stuck at 1 or 0). The PRI_REF and SEC_REF signals are sent to a stratum 3 or stratum 3E timing module 768. Timing module 768 includes an internal MUX for selecting between the PRI_REF and SEC_REF signals, and the timing module receives control and status signals 769 from the hardware control logic indicating whether PRI_REF or SEC_REF should be used. If one of the activity detectors 766a or 766b indicates an inactive status to the hardware control logic, then the hardware control logic sends appropriate information over control and status signals 769 to cause the timing module to select the active one of PRI_REF or SEC_REF.

The timing module also includes an internal phase locked loop (PLL) circuit and an internal stratum 3 or 3E oscillator. The timing module synchronizes its output signal 770 to the selected input signal (PRI_REF or SEC_REF). The timing module may be an MSTM-S3 available from Conner-Winfield or an ATIMe-s or ATIMe-3E available from TF systems. The hardware control logic, activity detectors and dual MUXs may be implemented in an FPGA. The timing module also includes a Free-run mode and a Hold-Over mode. When there is no input signal to synchronize to, the timing module enter a free-run mode and uses the internal oscillator to generate a clock output signal. If the signal being synchronized to is lost, then the timing module enters a hold-over

mode and maintains the frequency of the last known clock output signal for a period of time.

The EX CTS 750 also receives an external timing reference signal from the other EX CTS on STRAT_SYNC 755 (one of STRAT_REF1-STRAT_REFN from the other EX CTS). STRAT_SYNC and output 770 from the timing module are sent to a MUX 772a. REF_SEL(1:0) selection signals are sent from the hardware control logic to MUX 772a to select STRAT_SYNC when the EX CTS is the slave and output 770 when the EX CTS is the master. When in a test mode, the hardware control logic may also select a test input from a test header 771a.

An activity detector 774a monitors the status of output 770 from the timing module and provides a status signal to the hardware control logic. Similarly, an activity detector 774b monitors the status of STRAT_SYNC and provides a status signal to the hardware control logic. When the EX CTS is master, if the hardware control logic receives an inactive status from activity detector 774a, then the hardware control logic automatically changes the REF_SEL signals to select STRAT_SYNC forcing the EX CTS to switch over and become the slave. When the EX CTS is slave, if the hardware control logic receives an inactive status from activity detector 774b, then the hardware control logic may automatically change the REF_SEL signals to select output 770 from the timing module forcing the EX CTS to switch over and become master.

A MUX 772b receives feedback signals from the EX CTS itself. BENCH_FB is an external timing reference signal from the EX CTS that is routed back to the MUX on the local printed circuit board. STRAT_FB 754 is an external timing reference signal from the EX CTS (one of STRAT_REF1-STRAT_REFN) that is routed onto the mid-plane(s) and back onto the local printed circuit board such that is most closely resembles the external timing reference signals sent to the EX LTSs and the other EX CTS in order to minimize skew. The hardware control logic sends FB_SEL(1:0) signals to MUX 772b to select STRAT_FB in regular use or BENCH_FB or an input from a test header 771b in test mode.

The outputs of both MUX 772a and 772b are provided to a phase detector 776. The phase detector compares the rising edge of the two input signals to determine the magnitude of any phase shift between the two. The phase detector then generates variable voltage pulses on outputs 777a and 777b representing the magnitude of the phase shift. The phase detector outputs are used by discrete logic circuit 778 to generate a voltage on signal 779 representing the magnitude of the phase shift. The voltage is used to speed up or slow down (i.e., change the phase of) a VCXO 780 to allow the output signal 781 to track any phase change in the external timing reference signal received from the other EX CTS (i.e., STRAT_SYNC) or to allow the output signal 781 to track any phase change in the output signal 770 from the timing module. The discrete logic components implement a filter that determines how quickly or slowly the VCXO's output tracks the change in phase detected on the reference signal.

The phase detector circuit may be implemented in a programmable logic device (PLD).

The output 781 of the VCXO is sent to an External Reference Clock (ERC) circuit 782 which may also be implemented in a PLD. ERC_STRAT_SYNC is also sent to ERC 782 from the output of MUX 772a. When the EX CTS is the master, the ERC circuit generates the external timing reference signal 784 with an embedded processor timing reference signal, as described below, based on the output signal 781 and synchronous with ERC_STRAT_SYNC (corresponding to timing module output 770). When the EX CTS is the slave, the ERC generates the external timing reference signal 784 based on the output signal 781 and synchronous with ERC_STRAT_SYNC (corresponding to STRAT_SYNC 755 from the other EX CTS).

External reference signal 784 is then sent to a first level clock driver 785 and from there to second level clock drivers 786a-786d which provide external timing reference signals (STRAT_REF1-STRAT_REFN) that are distributed across the mid-plane(s) to EX LTSs on the other network device cards and the EX LTS on the same network device card, the other EX CTS and the EX CTS itself. The ERC circuit also generates

BITS1_TXREF and BITS2_TXREF signals that are provided to BITS T1/E1 framer 758.

The hardware control logic also includes an activity detector 788 that receives STRAT_REF_ACTIVITY from clock driver 785. Activity detector 788 sends a status signal to the hardware control logic, and if the status indicates that STRAT_REF_ACTIVITY is inactive, then the hardware control logic asserts KILL_CLKTREE. Whenever KILL_CLKTREE is asserted, the activity detector 774b in the other EX CTS detects inactivity on STRAT_SYNC and may become the master by selecting the output of the timing module as the input to MUX 772a.

Similar to hardware control logic 684 (Fig. 45) within the switch fabric CTS, hardware control logic 760 within the EX CTS implements a state machine (similar to the state machine shown in Fig. 46) based on IM_THE_MASTER and YOU_THE_MASTER signals sent between the two EX CTSs and also on slot identification signals (not shown).

In one embodiment, ports (e.g., 571a-571n, Fig. 49) on network device 540 are connected to external optical fibers carrying signals in accordance with the synchronous optical network (SONET) protocol and the external timing reference signal is a 19.44MHz signal that may be used as the SONET transmit reference clock. This signal may also be divided down to provide an 8KHz SONET framing pulse (i.e., J0FP) or multiplied up to provide higher frequency signals. For example, four times 19.44MHz is 77.76MHz which is the base frequency for a SONET OC1 stream, two times 77.76MHz provides the base frequency for an OC3 stream and eight times 77.76MHz provides the base frequency for an OC12 stream.

In one embodiment, the embedded processor timing reference signal within the 19.44MHz external timing reference signal is 8KHz. Since the processor timing reference signal and the SONET framing pulse are both 8KHz, the embedded processor timing reference signal may used to supply both. In addition, the embedded processor timing reference signal may also be used to supply BITS1_TXREF and BITS2_TXREF signals to BITS T1/E1 framer 758.

Referring to Fig. 51, the 19.44MHz external reference timing signal with embedded 8KHz processor timing reference signal from ERC 782 (i.e., output signal 784) includes a duty-cycle distortion 790 every 125 microseconds (us) representing the embedded 8KHz signal. In this embodiment, VCXO 780 is a 77.76 MHz VCXO providing a 77.76 MHz clock output signal 781. The ERC uses VCXO output signal 781 to generate output signal 784 as described in more detail below. Basically, every 125us, the ERC holds the output signal 784 high for one extra 77.76MHz clock cycle to create a 75% / 25% duty cycle in output signal 784. This duty cycle distortion is used by the EX LTSs and EX CTSs to extract the 8KHz signal from output signal 784, and since the EX LTS's use only the rising edge of the 19.44MHz signal to synchronize local external timing signals, the duty cycle distortion does not affect that synchronization.

External Reference Clock (ERC) circuit:

Referring to Fig. 52, an embeddor circuit 792 within the ERC receives VCXO output signal 781 (77.76MHz) at four embedding registers 794a-794d, a 9720-1 rollover counter 796 and three 8KHz output registers 798a-798b. Each embedding register passes its value (logic 1 or 0) to the next embedding register, and embedding register 794d provides ERC output signal 784 (19.44MHz external timing reference signal with embedded 8KHz processor timing reference signal). The output of embedding register 794b is also inverted and provided as an input to embedding register 794a. When running, therefore, the embedding registers maintain a repetitive output 784 of a high for two 77.76MHz clock pulses and then low for two 77.76MHz which provides a 19.44MHz signal. Rollover counter 796 and a load circuit 800 are used to embed the 8KHz signal.

The rollover counter increments on each 77.76MHz clock tick and at 9720-1 (9720-1 times 77.76MHz = 8KHz), the counter rolls over to zero. Load circuit 800 detects when the counter value is zero and loads a logic 1 into embedding registers 794a, 794b and 794c and a logic zero into embedding register 794d. As a result, the output of embedding register 794d is held high for three 77.76MHz clock pulses (since logic ones are loaded into three embedding registers) which forces the duty cycle distortion into the 19.44MHz output signal 784.

BITS circuits 802a and 802b also monitor the value of the rollover counter. While the value is less than or equal to 4860-1 (half of 8KHz), the BITS circuits provide a logic one to 8KHz output registers 798a and 798b, respectively. When the value changes to 4860, the BITS circuits toggle from a logic one to a logic zero and continue to send a logic zero to 8KHz output registers 798a and 798b, respectively, until the rollover counter rolls over. As a result, 8KHz output registers 798a and 798b provide 8KHz signals with a 50% duty cycle on BITS1_TXREF and BITS2_TXREF to the BITS T1/E1 framer.

As long as a clock signal is received over signal 781 (77.76MHz), rollover counter 796 continues to count causing BITS circuits 802a and 802b to continue toggling 8KHz registers 798a and 798b and causing load circuit 800 to continue to load logic 1110 into the embedding registers every 8KHz. As a result, the embedding registers will continue to provide a 19MHz clock signal with an embedded 8KHz signal on line 784. This is often referred to as "fly wheeling."

Referring to Fig. 53, an extractor circuit 804 within the ERC is used to extract the embedded 8 KHz signal from ERC_STRAT_SYNC. When the EX CTS is the master, ERC_STRAT_SYNC corresponds to the output signal 770 from the timing module 768 (pure 19.44MHz), and thus, no embedded 8KHz signal is extracted. When the EX CTS is the slave, ERC_STRAT_SYNC corresponds to the external timing reference signal provided by the other EX CTS (i.e., STRAT_SYNC 755; 19.44MHz with embedded 8KHz) and the embedded 8KHz signal is extracted. The extractor circuit includes three extractor registers 806a-806c. Each extractor register is connected to the 77.76MHz VCXO output signal 781, and on each clock pulse, extractor register 806a receives a logic one input and passes its value to extractor register 806b which passes its value to extractor register 806c which provides an 8KHz pulse 808. The extractor registers are also connected to ERC_SRAT_SYNC which provides an asynchronous reset to the extractor registers – that is, when ERC_STRAT_SYNC is logic zero, the registers are reset to zero. Every two 77.76MHz clock pulses, therefore, the extractor registers are reset and for most cycles, extractor register 806c passes a logic zero to output signal 808. However, when the EX CTS is the slave, every 8KHz ERC_STRAT_SYNC remains a logic one

for three 77.76 MHz clock pulses allowing a logic one to be passed through each register and onto output signal 808 to provide an 8KHz pulse.

8KHz output signal 808 is passed to extractor circuit 804 and used to reset the rollover counter to synchronize the rollover counter to the embedded 8KHz signal within ERC_STRAT_SYNC when the EX CTS is the slave. As a result, the 8KHz embedded signal generated by both EX CTSs are synchronized.

External Local Timing Subsystem (EX LTS):

Referring to Fig. 54, EX LTS 756 receives STRAT_REF_B from one EX CTS and STRAT_REF_A from the other EX CTS. STRAT_REF_B and STRAT_REF_A correspond to one of STRAT_REF1-STRAT_REFN (Fig. 50) output from each EX CTS. STRAT_REF_B and STRAT_REF_A are provided as inputs to a MUX 810a and a hardware control logic 812 within the EX LTS selects the input to MUX 810a using REF_SEL (1:0) signals. An activity detector 814a monitors the activity of STRAT_REF_A and sends a signal to hardware control logic 812 if it detects an inactive signal (i.e., stuck at logic one or zero). Similarly, an activity detector 814b monitors the activity of STRAT_REF_B and sends a signal to hardware control logic 812 if it detects an inactive signal (i.e., stuck at logic one or zero). If the hardware control logic receives a signal from either activity detector indicating that the monitored signal is inactive, the hardware control logic automatically changes the REF_SEL (1:0) signals to cause MUX 810a to select the other input signal and send an interrupt to the local processor.

A second MUX 810b receives a feed back signal 816 from the EX LTS itself. Hardware control logic 812 uses FB_SEL(1:0) to select either a feedback signal input to MUX 810b or a test header 818b input to MUX 810b. The test header input is only used in a test mode. In regular use, feedback signal 816 is selected. Similarly, in a test mode, the hardware control logic may use REF_SEL(1:0) to select a test header 818a input to MUX 810a.

Output signals 820a and 820b from MUXs 810a and 810b, respectively, are provided to phase detector 822. The phase detector compares the rising edge of the two input signals to determine the magnitude of any phase shift between the two. The phase

detector then generates variable voltage pulses on outputs 821a and 821b representing the magnitude of the phase shift. The phase detector outputs are used by discrete logic circuit 822 to generate a voltage on signal 823 representing the magnitude of the phase shift. The voltage is used to speed up or slow down (i.e., change the phase of) of an output 825 of a VCXO 824 to track any phase change in STRAT_REF_A or STRAT_REF_B. The discrete logic components implement filters that determine how quickly or slowly the VCXO's output will track the change in phase detected on the reference signal.

In one embodiment, the VCXO is a 155.51MHz or a 622MHz VCXO. This value is dependent upon the clock speeds required by components, outside the EX LTS but on the local card, that are responsible for transferring network data over the optical fibers in accordance with the SONET protocol. On at least the universal port card, the VCXO output 825 signal is sent to a clock driver 830 for providing local data transfer components with a 622MHz or 155.52MHz clock signal 831.

The VCXO output 825 is also sent to a divider chip 826 for dividing the signal down and outputting a 77.76MHz output signal 827 to a clock driver chip 828. Clock driver chip 828 provides 77.76MHz output signals 829a for use by components on the local printed circuit board and provides 77.76MHz output signal 829b to ERC circuit 782. The ERC circuit also receives input signal 832 corresponding to the EX LTS selected input signal either STRAT_REF_B or STRAT_REF_A. As shown, the same ERC circuit that is used in the EX CTS may be used in the EX LTS to extract an 8KHz J0FP pulse for use by data transfer components on the local printed circuit board. Alternatively, the ERC circuit could include only a portion of the logic in ERC circuit 782 on the EX CTS.

Similar to hardware control logic 712 (Fig. 47) within the switch fabric LTS, hardware control logic 812 within the EX LTS implements a state machine (similar to the state machine shown in Fig. 48) based on signals from activity detectors 814a and 814b.

External Reference Clock (ERC) circuit:

Referring again to Figs. 52 and 53, when the ERC circuit is within an EX LTS circuit, the inputs to extractor circuit 804 are input signal 832 corresponding to the LTS selected input signal either STRAT_REF_B or STRAT_REF_A and 77.76MHz clock input signal 829b. The extracted 8KHz pulse 808 is again provided to embeddor circuit 792 and used to reset rollover counter 796 in order to synchronize the counter with the embedded 8KHz signal with STRAT_REF_A or STRAT_REF_B. Because the EX CTSs that provide STRAT_REF_A and STRAT_REF_B are synchronous, the embedded 8KHz signals within both signals are also synchronous. Within the EX LTS, the embedding registers 794a-794d and BITS registers 798a and 798b are not used. Instead, a circuit 834 monitors the value of the rollover counter and when the rollover counter rolls over to a value of zero, circuit 834 sends a logic one to 8KHz register 798c which provides an 8KHz pulse signal 836 that may be sent by the LTS to local data transfer components (i.e., J0FP) and processor components as a local processor timing signal.

Again, as long as a clock signal is received over signal 829b (77.76MHz), rollover counter 796 continues to count causing circuit 834 to continue pulsing 8KHz register 798c.

External Central Timing Subsystem (EX CTS) Alternate Embodiment:

Referring to Fig. 55, instead of using one of the STRAT_REF1-STRAT_REFN signals from the other EX CTS as an input to MUX 772a, the output 770 (marked "Alt. Output to other EX CTS") of timing module 768 may be provided to the other EX CTS and received as input 838 (marked "Alt. Input from other EX CTS"). The PLL circuit, including MUXs 772a and 772b, phase detector 776, discrete logic circuit 778 and VCXO 780, is necessary to synchronize the output of the VCXO with either output 770 of the timing module or a signal from the other EX CTS. However, PLL circuits may introduce jitter into their output signals (e.g., output 781), and passing the PLL output signal 781 via one of the STRAT_REF1-STRAT_REFN signals from one EX CTS into the PLL of the other EX CTS – that is, PLL to PLL -- may introduce additional jitter into output signal 781. Since accurate timing signals are critical for proper data transfer with other network devices and SONET standards specifically set maximum allowable jitter transmission at interfaces (Bellcore GR-253-CORE and SONET Transport Systems Common Carrier Criteria), jitter should be

minimized. Passing the output 770 of the timing module within the EX CTS to the input 838 of the other EX CTS avoids passing the output of one PLL to the input of the second PLL and thereby reduces the potential introduction of jitter.

It is still necessary to send one of the STRAT_REF1-STRAT_REFN signals to the other EX CTS (received as STRAT_SYNC 755) in order to provide ERC 782 with a 19.44MHz signal with an embedded 8KHz clock for use when the EX CTS is a slave. The ERC circuit only uses ERC_STRAT_SYNC in this instance when the EX CTS is the slave.

Layer One Test Port:

The present invention provides programmable physical layer (i.e., layer one) test ports within an upper layer network device (e.g., network device 540, Fig. 35). The test ports may be connected to external test equipment (e.g., an analyzer) to passively monitor data being received by and transmitted from the network device or to actively drive data to the network device. Importantly, data provided at a test port accurately reflects data received by or transmitted by the network device with minimal modification and no upper layer translation or processing. Moreover, data is supplied to the test ports without disrupting or slowing the service provided by the network device.

Referring to Figs. 35 and 36, network device 540 includes at least one cross-connection card 562a-562b, 564a-564b, 566a-566b, 568a-568b, at least one universal port card 554a-554h, 556a-556h, 558a-558h, 560a-560h, and at least one forwarding card 546a-546e, 548a-548e, 550a-550e, 552a-552e. Each port card includes at least one port 571a-571n for connecting to external physical network attachments 576a-576b, and each port card transfers data to a cross-connection card. The cross-connection card transfers data between port cards and forwarding cards and between port cards. In one embodiment, each forwarding card includes at least one port/payload extractor 582a-582n for receiving data from the cross-connection cards.

Referring to Fig. 56, a port 571a on a port card 554a within network device 540 may be connected to another network device (not shown) through physical external network attachments 576a and 576b. As described above, components 573 on the

port card transfer data between port 571a and cross-connection card 562a, and components 563 on the cross-connection card transfer data on particular paths between the port cards and the forwarding cards or between port cards. For convenience, only one port card, forwarding card and cross-connection card are shown.

For many reasons, including error diagnosis, a service administrator may wish to monitor the data received on a particular path or paths at a particular port, for example, port 571a, and/or the data transmitted on a particular path or paths from port 571a. To accomplish this, the network administrator may connect test equipment, for example, an analyzer 840 (e.g., an Omniber analyzer available from Hewlett Packard Company), to the transmit connection of port 571b to monitor data received at port 571a and/or to the transmit connection of port 571c to monitor data transmitted from port 571a. The network administrator then notifies the NMS (e.g., NMS 60 running on PC 62, Fig. 35) as to which port or ports on which port card or port cards should be enabled and whether the transmitter and/or receiver for each port should be enabled. The network administrator also notifies the NMS as to which path or paths are to be sent to each test port, and the time slot for each path. With this information, the NMS fills in test path table 841 (Figs. 57 and 58) in configuration database 42.

Similar to the process of enabling a working port through path table 600 (Figs. 37 and 38), when a record in the test path table is filled in, the configuration database sends an active query notification to the path manager (e.g., path manager 597) executing on the universal port card (e.g., port card 554a) corresponding to the universal port card port LID in the path table record. For example, port 571b may have a port LID of 1232 (record 842, Fig. 58) and port 571b may have a port LID of 1233 (record 843). An active query notification is also sent to NMS database 61, and once the NMS database is updated, the NMS displays the new system configuration, including the test ports, to the user.

Through the test path table, the path manager learns that the transmitters of ports 571b and 571c need to be enabled and which path or paths are to be transferred to each port. As shown in path table 600 (Fig. 38), path LID 1666 corresponds to working port LID 1231 (port 571a), and as shown in test path table 841 (Fig. 58), path LID

1666 is also assigned to test port LIDs 1232 and 1233 (ports 571b and 571c, respectively). Record 842 indicates that the receive portion of path 1666 (i.e., "ingress" in Monitor column 844) is to be sent to port LID 1232 (i.e., port 571b) and then transmitted (i.e., "no" in Enable Port Receiver column 845) from port LID 1232, and similarly, record 843 indicates that the transmit portion of path 1666 (i.e., "egress" in Monitor column 844) is to be sent to port LID 1233 (i.e., port 571c) and then transmitted (i.e., "no" in Enable Port Receiver column 845) from port LID 1233.

The path manager passes the path connection information to cross-connection manager 605 executing on the cross-connection card 562a. The CCM uses the connection information to generate a new connection program table 601 and uses this table to program internal connections through one or more components (e.g., a TSE chip 563) on the cross-connection card. After re-programming, cross-connection card 562a continues to transmit data corresponding to path LID 1666 between port 571a on universal port card 554a and the serial line input to payload extractor 582a on forwarding card 546c. However, after reprogramming, cross-connection card 562a also multicasts the data corresponding to path LID 1666 and received on port 571a to port 571b and data corresponding to path LID 1666 and transmitted to port 571a by forwarding card 546c to port 571c.

Analyzer 840 may then be used to monitor both the network data received on port 571a and the network data being transmitted from port 571a. Alternatively, analyzer 840 may only be connected to one test port to monitor either the data received on port 571a or the data transmitted from port 571a. The data received on port 571a may be altered by the components on the port card(s) and the cross-connection cards before the data reaches the test port but any modification is minimal. For example, where the external network attachment 576a is a SONET optical fiber, the port card components may convert the optical signals into electrical signals that are passed to the cross-connection card and then back to the test ports, which reconvert the electrical signals into optical signals before the signals are passed to analyzer 840. Since the data received at port 571a has not been processed or translated by the upper layer processing components on the forwarding card, the data accurately reflects the data received at the port. For example, the physical layer (e.g., SONET) information and format is accurately reflected in the data received.

To passively monitor both the data received and transmitted by a particular port, two transmitters are necessary and, thus, two ports are consumed for testing and cannot be used for normal data transfer. Because the test ports are programmable through the cross-connection card, however, the test ports may be re-programmed at any time to be used for normal data transfer. In addition, redundant ports may be used as test ports to avoid consuming ports needed for normal data transfer. Current network devices often have a dedicated test port that can provide both the data received and transmitted by a working port. The dedicated test port, however, contains specialized hardware that is different from the working ports and, thus, cannot be used as a working port. Hence, although two ports may be consumed for monitoring the input and output of one working port, they are only temporarily consumed and may be re-programmed at any time. Similarly, if the port card on which a test port is located fails, the test port(s) may be quickly and easily reprogrammed to another port on another port card that has not failed.

Instead of passively monitoring the data received at port 571a, test equipment 840 may be connected to the receiver of a test port and used to drive data to network device 540. For example, the network administrator may connect test equipment 840 to the receiver of test port 571c and then notify the NMS to enable the receiver on port 571c to receive path 1666. With this information, the NMS modifies test path table 841. For example, record 844 (Fig. 58) indicates that the receive portion of path 1666 (i.e., "ingress" in Monitor column 844) is to be driven (i.e., "yes" in Enable Port Receiver column 845) externally with data from port LID 1233 (i.e., port 571c). Again, an active query notification is sent to path manager 597. Path manager 597 then disables the receiver corresponding to port LID 1231 (i.e., port 571a) and enables the receiver corresponding to port LID 1233 (i.e., port 571c) and passes the path connection information to cross-connection manager 605 indicating that port LID 1231 will supply the receive portion of path 1666. The cross-connection manager uses the connection information to generate a new connection program table 601 to re-program the internal connections through the cross-connection card. In addition, the network administrator may also indicate that the transmitter of port 571a should be disabled, and path manager 597 would disable the transmitter of port 571a and pass the connection information to the cross connection manager.

After re-programming, cross-connection card 562a data is sent from test equipment 840 to test port 571c and then through the cross-connection card to forwarding card 546c. The cross-connection card may multicast the data from forwarding card 546c to both working port 571a and to test port 571c, or just to test port 571c or just working port 571a.

Instead of having test equipment 840 drive data to the network device over a test port, internal components on a port card, cross-connection card or forwarding card within the network device may drive data to the other cards and to other network devices over external physical attachments connected to working ports and/or test ports. For example, the internal components may be capable of generating a pseudo-random bit sequence (PRBS). Test equipment 840 connected to one or more test ports may then be used to passively monitor the data sent from and/or received by the working port, and the internal components may be capable of detecting a PRBS over the working port and/or test port(s).

Although the test ports have been shown on the same port card as the working port being tested, it should be understood, that the test ports may be on any port card in the same quadrant as the working port. Where cross-connection cards are interconnected, the test ports may be on any port card in a different quadrant so long as the cross-connection card in the different quadrant is connected to the cross-connection card in same quadrant as the working port. Similarly, the test ports may be located on different port cards with respect to each other. A different working port may be tested by re-programming the cross-connection card to multicast data corresponding to the different working port to the test port(s). In addition, multiple working ports may be tested simultaneously by re-programming the cross-connection card to multicast data from different paths on different working ports to the same test port(s) or to multiple different test ports. A network administrator may choose to dedicate certain ports as test ports prior to any testing needing to be done or the network administrator may choose certain ports as test ports when problems arise.

The programmable physical layer test port or ports allow a network administrator to test data received at or transmitted from any working port or ports and also to drive

data to any upper layer card (i.e., forwarding card) within the network device. Only the port card(s) and cross-connection card need be working properly to passively monitor data received at and sent from a working port. Testing and re-programming test ports may take place during normal operation without disrupting data transfer through the network device to allow for diagnosis without network device disruption.

NMS Server Scalability

As described above, a network device (e.g., 10, Fig. 1 and 540, Fig. 35) may include a large number (e.g., millions) of configurable / manageable objects. Manageable objects are typically considered physical or logical. Physical managed objects correspond to the physical components of the network device such as the network device itself, one or more chassis within the network device, shelves in each chassis, slots in each shelf, cards inserted in each slot, physical ports on particular cards (e.g., universal port cards), etc. Logical managed objects correspond to configured elements of the network device such as SONET paths, internal logical ports (e.g., forwarding card ports), ATM interfaces, virtual ATM interfaces, virtual connections, paths/interfaces related to other network protocols (e.g., MPLS, IP, Frame Relay, Ethernet), etc.

If multiple NMS clients request access to multiple different network devices and the NMS server is required to retrieve and store data for all managed objects corresponding to each network device, then the NMS server's local memory will likely be quickly filled and repeated retrievals of data from each network device will likely be necessary. Retrieval of a large amount of data from each network device limits the scalability of the NMS server and reduces the NMS server's response time to NMS client requests.

To improve the scalability of the NMS server and improve data request response times, only physical managed objects are initially retrieved from a selected network device and logical managed objects are retrieved only when necessary. To further increase NMS server scalability and response time, proxies for managed objects (preferably physical managed objects and only a limited number of global logical managed objects) are stored in memory local to each NMS client. Moreover, to increase NMS server scalability and response time, unique identification numbers

corresponding to each managed object are also stored in memory local to the NMS client (for example, in proxies or GUI tables) and used by the NMS server to quickly retrieve data requested by the NMS client. Each NMS client, therefore, maintains its user context of interest, eliminating the need for client-specific device context management by the NMS server.

Referring to Fig. 59, an NMS client 850a runs on a personal computer or workstation 984 and uses data in graphical user interface (GUI) tables 985 stored in local memory 986 to display a GUI to a user (e.g., network administrator, provisioner, customer) after the user has logged in. In one embodiment, the GUI is GUI 895 described above with reference to Figs. 4a-4z, 5a-5z, 6a-6p, 7a-7y, 8a-8e, 9a-9n, 10a-10i and 11a-11g. When GUI 895 is initially displayed (see Fig. 4a), only navigation tree 898 is displayed and under Device branch 898a a list 898b of IP addresses and/or domain name server (DNS) names may be displayed corresponding to network devices that may be managed by the user in accordance with the user's profile.

If the user selects one of the IP addresses (e.g., 192.168.9.202, Fig. 4f) in list 898b, then the client checks local memory 986 (Fig. 59) for proxies (described below) corresponding to the selected network device and if such proxies are not in local memory 986, the NMS client sends a network device access request including the IP address of the selected network device to an NMS server, for example, NMS server 851a. The NMS server may be executed on the same computer or workstation as the client or, more likely, on a separate computer 987. The NMS server checks local memory 987a for managed objects corresponding to the network device to be accessed and if the managed objects are not in local memory 987a, the NMS server sends database access commands to the configuration database 42 within the network device corresponding to the IP address sent by the NMS client. The database access commands retrieve only data corresponding to physical components of the network device.

In one embodiment, data is stored within configuration database 42 as a series of containers. Since the configuration database is a relational database, data is stored in tables and containment is accomplished using pointers from lower level tables (children) to upper level tables (parents). As previously discussed with reference to

Figs. 12a-12c, after the network device is powered-up, the Master MCD (Master Control Driver) 38 takes a physical inventory of the network device (e.g., computer system 10, Fig. 1, network device 540, Figs. 35, 59) and assigns a unique physical identification number (PID) to each physical component within the system, including the network device itself, each chassis in the network device, each shelf in each chassis, each slot in each shelf, each card inserted in each slot, and each port on each card having a physical port (e.g., universal port cards). As previously stated, the PID is a unique logical number unrelated to any physical aspect of the component.

The MCD then fills in tables for each type of physical component, such tables being provided by a default configuration within the configuration database. Alternatively, the MCD could create and fill in each table. In one embodiment, the configuration database includes a managed device table 983 (Fig. 60a), a chassis table 988 (Fig. 60b), a shelf table 989 (Fig. 60c), a slot table 990 (Fig. 60d), a card table 47' (Fig. 60e), and a port table 49' (Fig. 60f). The MCD enters the assigned unique PID for each physical component in a row (i.e., record) in one of the tables. Consequently, each unique PID serves as a primary key within the configuration database for the row / data corresponding to each physical component. Where available, the MCD also enters data representing attributes (e.g., card type, port type, relative location, version number, etc.) for the component in each table row. In addition, with the exception of the managed device table, each row includes a unique PID corresponding to a parent table. The unique PID corresponding to a parent table is a pointer and provides data "containment" by linking each child table to its parent table (i.e., provides a table hierarchy). The unique PID corresponding to the parent table may also be referred to as a foreign key for association.

Referring to Fig. 60a, since the managed device is the top physical level, managed device table 983 includes one row 983a representing the one managed device (e.g., 540, Figs. 35 and 59) including a unique managed device PID 983b (e.g., 1; i.e., primary key) and attributes A1-An corresponding to the managed device but the managed device table does not include a parent PID (i.e., foreign key for association). In the current embodiment, chassis table 988 includes one row 988a representing the one chassis (e.g., 620, Figs. 41a-41b) in the managed device. Other network devices may have multiple chassis and a row would be added to the chassis table for each

chassis and each row would include the same managed device PID (e.g., 1). Each row in the chassis table includes a unique chassis PID 988b (e.g., 2; i.e., primary key) and attributes A1-An corresponding to the chassis and a managed device PID 988c (i.e., parent PID / foreign key for association). Referring to Fig. 60c, shelf table 989 includes one row for each shelf in the chassis and each row includes a unique shelf PID 989a (e.g., 3-18; i.e., primary key) and attributes A1-An corresponding to each shelf and a chassis PID 989b (i.e., foreign key for association). Since all the shelves are in the same chassis in this embodiment, they each list the same chassis PID (e.g., 2). Referring to Fig. 60d, slot table 990 includes one row for each slot in the chassis and each row includes a unique slot PID 990a (e.g., 20-116; i.e., primary key) and attributes A1-An corresponding to each slot and a shelf PID 990b (i.e., foreign key for association). Since there may be many shelves in the chassis, the shelf PID in each row corresponds to the shelf in which the slot is located. For example, a row 990c includes slot PID 20 corresponding to a shelf PID of 3, and a row 990d includes slot PID 116 corresponding to a different shelf PID of 18.

Referring to Fig. 60e, card table 47' includes one row for each card inserted within a slot in the chassis and each row includes a unique card PID 47a (i.e., primary key), attributes (e.g., CWD Type, Version No., etc.) corresponding to each card and a slot PID 47b (i.e., foreign key for association) corresponding to the slot in which the card is inserted. Referring to Fig. 60f, port table 49' includes one row for each physical port located on a universal port card in the chassis and each row includes a unique port PID 49a (i.e., primary key), attributes (e.g., port type, version no., etc.) corresponding to each port and a card PID 49b (i.e., foreign key for association) corresponding to the card on which the port is located.

Even after initial power-up, master MCD 38 continues to take physical inventories of the network device to determine if physical components have been added or removed. For example, cards may be added to empty slots or removed from slots. When changes are detected, master MCD 38 updates the tables (e.g., card table 47' and port table 49') accordingly, and through the active query feature, the configuration database updates an external NMS database (e.g., 61, Fig. 59) and notifies the NMS server. In one embodiment, each time a physical component is changed, the NMS server sends the NMS client a full set of updated proxies to ensure that the NMS

client is fully synchronized with the network device. Alternatively, only those proxies that are affected may be updated. As described below, however, proxies may include pointers to both a parent proxy and children proxies, and if so, even a change to only one physical component requires changes to the proxy for that component and any related parent and/or children proxies.

In this embodiment, therefore, when the server sends database access commands to the configuration database within the network device to retrieve all data corresponding to physical components of the network device, the database access commands request data from each row in each of the physical tables (e.g., managed device table 983, chassis table 988, shelf table 989, slot table 990, card table 47' and port table 49'). The data from these tables is then sent to the NMS server, and the server creates physical managed objects (PMO1-PMOn, Fig. 59) for each row in each table and stores them in local memory 987a.

Referring to Fig. 61a, each physical managed object 991 created by the NMS server includes the unique PID 991a and the attribute data 991b associated with the particular row / record in the configuration database table and function calls 991c. With the exception of the managed device physical managed object, the attribute data includes a pointer (i.e., PID) for the corresponding parent physical component, and with the exception of the port physical managed objects, each managed object's attribute data also includes one or more pointers (i.e., PIDs) corresponding to any children physical components. In this embodiment, the port managed objects are the lowest level physical component and, therefore, do not include pointers to children physical components.

In one embodiment, all physical managed objects include a "Get Parent" 991e function call to cause the NMS server to retrieve data corresponding to the parent physical component. A Get Parent function call to the managed device managed object receives a null message since the managed device does not have a parent component. The Get Parent function call may be used for constraint checking. For example, prior to configuring a particular card as a backup for another card, the Get Parent function call may be placed twice by the NMS server to ensure that both cards are within the same shelf – that is, the network device may have a constraint that

redundant boards must be within the same shelf. The first Get Parent function call determines which slot each card is in and the second Get Parent function call determines which shelf each slot is in. If the shelves match, then the constraint is met.

In one embodiment, all physical managed objects include a "Get Children" 991f function call to cause the NMS server to retrieve data from the configuration database for children physical components related to the physical managed object. A Get Children function call to a port managed object receives a null message since the port does not have any physical children components. The data retrieved with the Get Children function call is used to fill in the tables in the physical tabs (e.g., system tab 934 (Fig. 4s), module tab 936 (Fig. 4t), ports tab 938 (Fig. 4u) and SONET Interfaces tab 940 (Fig. 4v)) within configuration / status window 897 (Fig. 5q). Some or all of the data from each row in the configuration database tables may be used to fill in these tables.

In addition to Get Children and Get Parent function calls, each physical managed object includes a "Get Config" 991g and a "Set Config" 991h function call. The Get Config function call is used to retrieve data for dialog boxes when a user double clicks the left mouse button on an entry in one of the tabs in status window 897. The Set Config function call is used to implement changes to managed objects received from a user through a dialog box.

Instead of a "Get Children" function call, the port managed object includes a "Get SONET Path Table" function call to cause the server to retrieve all SONET paths (logical managed objects) configured for that particular port for display in SONET Paths tab 942 (Fig. 5q). Since SONET paths are children to a port, the "Get SONET Path Table" corresponds to the "Get Children" function call in the other physical managed objects. However, the pointers (i.e., logical identification numbers (LIDs)) to the children are not stored in the port managed object attribute data. This is because the number of SONET paths that the SONET port would need to point may be large and would have to be regularly updated as SONET Paths are created and deleted. The port managed object also includes a "Create SONET Path" function call and a "Delete SONET Path" function call to cause the server to create or delete,

respectively, a SONET path for that particular port. As described below, the port managed object may also include other function calls related to logical components.

Each managed object 991 also includes a "Get Proxy" function call 991d, and after creating each managed object, the NMS server places a get proxy function call to the managed object. Placing the get proxy call causes the NMS server to create a proxy (PX) for the managed object and send the proxy (e.g., PX1-PXn) to memory 986 local to the NMS client that requested the network device access. Referring to Fig. 61b, each proxy includes the PID 992a and some or all of the attribute data 992b from the corresponding managed object. The decision to include some or all of the attribute data within the proxy may depend upon the size of the memory 986 local to the NMS client. This may be a static design decision based on the expected size of the memory local to the typical NMS client, or this may be a dynamic decision based on the actual size of the memory local to the NMS client that requested access to the network device. If sufficiently large, the proxy may include all the attribute data. If not sufficiently large, then perhaps only attribute data regularly accessed by users may be included in the proxy. For example, for a port managed object perhaps only the port name, connection type and relative position within the network device is included in the proxy.

In addition, each proxy may include function calls 992c similar to one or more function calls in the corresponding managed object, with the exception of the "Get Proxy" function call. Unlike the managed object function calls, however, the proxy function calls cause the NMS client to send messages to the NMS server in, for example, JAVA RMI. For instance, the SONET Port proxy like the SONET Port managed object includes the "Get SONET Path Table", "Create SONET Paths" and "Delete SONET Paths" function calls. However, proxy function calls cause the NMS client to send JAVA RMI messages to the NMS server to cause the server to place similar function calls to the managed object. The managed object function calls cause the server to generate database access commands to the configuration database in the network device.

Initially, the NMS client uses data from the received proxies (PX1-PXn, Fig. 59) to update GUI tables 985 which causes the GUI to display device mimic 896a (Fig. 4f)

in graphic window 896b and system tab 934 (Fig. 4s) in configuration / service status window 897. Limiting the initial data retrieval from the configuration database to only data corresponding to physical components of the network device – as opposed to both physical and logical components -- reduces the amount of time required to transfer the data from the configuration database to the NMS server and on to the NMS client. Thus, the NMS client is able to display the device mimic and system tab more quickly than if data corresponding to both the physical and logical components were retrieved. To further increase the speed with which the device mimic and system tab are displayed, the NMS server may first transfer the proxies necessary for the device mimic and the system tab and then transfer the proxies corresponding to other physical tabs, including module (i.e., card) tab 936 (Fig. 4t), port tab 938 (Fig. 4u) and SONET Interfaces tab 940 (Fig. 4v).

If a user selects a different network device from navigation tree 898 (Fig. 5h) using NMS client 850a, NMS client 850a searches local memory 986 for proxies associated with the selected network device and if not found, the NMS client sends JAVA RMI messages to the NMS server to cause the NMS server to retrieve all physical data from the selected network device, create physical managed objects, store them in local memory 987a, create proxies for each physical managed object and send the proxies to the NMS client. If memory 986 local to the NMS client is sufficiently large, then the proxies for the first selected network device may remain in memory along with the proxies for the second selected network device. Consequently, if the user re-selects the first selected network device, the proxies are located in local memory by the NMS client, and the NMS client does not have to access the NMS server.

In addition to reducing the time required to display physical information through GUI 895, limiting the initial data retrieval to only physical data reduces the amount of memory 987a local to the NMS server required to store the managed objects. Moreover, once the data from the proxies are added to the GUI tables, the GUI can respond to a user request for any of the device views within the mimic (as shown in Fig.s 4f-4r) and to a user request for any physical tab without having to send data requests to the NMS server. Consequently, the GUI response time is increased, traffic between the NMS client and server is reduced and the burden on the server to respond to client requests is reduced.

If the proxies include all of the attribute data from the managed objects, then once the proxies are transferred to the NMS client, it is not necessary for the NMS server to continue storing the corresponding physical managed objects. If, however, a proxy includes only some of the attribute data from its corresponding managed object, then continuing to store the managed object at the NMS server saves time if the user requests access to data not included in the proxy. For example, a proxy may only include data for attributes displayed in a tab in status window 897. If a user desires more data, the user may double click the left mouse button on an entry in the tab to cause a dialog box to be displayed including additional attribute data. This causes the NMS client to place a Get Config function call to the corresponding proxy which causes the NMS client to send JAVA RMI messages to the NMS server. If the managed object is still in local memory 987a, then the response time to the client is faster than if the server needs to access the configuration database again to retrieve the data.

Maintaining the managed objects for a particular network device in local memory 987a is also advantageous if another NMS client requests access to the same network device. As previously mentioned, when the NMS server receives a network device access request, it first checks local memory 987a. If the managed objects are already present, then the NMS server may respond more quickly than if the server again needs to retrieve the data from the network device.

Due to the advantages described above, in one embodiment, the NMS server does not automatically delete managed objects from its local memory after proxies are sent to the NMS client. However, because the NMS server's local memory is a limited resource, as clients request access to more and more different network devices, it may become necessary for the NMS server to overwrite managed objects within local memory 987a such that they are no longer available. As previously mentioned, sending proxies to the NMS clients allows the clients to display physical data through GUI 895 without accessing the NMS server. Thus, even when the NMS server is forced to overwrite corresponding managed objects in local memory 987a, the client is able to continue displaying physical data through GUI 895.

Importantly, through the unique PID and the function calls, the proxies also provide an improved mechanism for accessing logical data and physical data not included within the proxies. As mentioned above, if the user requests access to physical data not in the proxy, then the NMS client places a Get Config function call to the NMS server. The function call is made more efficient by including the unique PID stored in the proxy. The NMS server uses the PID to first search local memory 987a – perhaps the NMS server searches a hash table in cache. If the PID is found, then the NMS quickly sends the data from the corresponding managed object to the NMS client. If the PID is not found in local memory 987a, then the NMS server uses the PID as a primary key to retrieve the physical data from the configuration database within the network device and again builds the corresponding physical managed object. The NMS server then sends the data from the managed object to the NMS client.

Without the PID, the NMS server would be forced to walk through the hierarchical physical tables until the correct physical component was found. For example, if the NMS server needs data relevant to a particular port, the NMS server would begin by locating the managed device, the chassis, then the correct shelf within the chassis, then the correct slot within the chassis, then the module within the slot and then finally the correct port on the module. This will likely take several database accesses and will certainly take more time than directly accessing the port data using a primary key that provides absolute context.

The process is similar if the data requested is logical. For example, if a user selects a particular port (e.g., port 939a, Fig. 5a) and then selects SONET Paths tab 942 (Fig. 5h), the logical data associated with the SONET paths configured for the selected port (e.g., SONET paths 942a and 942b) is needed. To do this, the NMS client places a "Get SONET Path Table" function call to the port proxy which causes the NMS client to issue JAVA RMI messages to the NMS server including a request for the SONET paths configured for the physical port associated with the unique port PID stored in the proxy. The NMS server first searches local memory 987a for the PID. If a managed object including the PID is found in local memory, then the NMS server places a similar "Get SONET Path Table" function call through the port managed object. If the PID is not found in local memory, then the NMS server uses the port PID as a primary key to quickly retrieve the data from the configuration database

stored in the table row corresponding to the selected port. The NMS server again builds the managed object for the port and then places the "Get SONET Path Table" function call through the managed object. The Get SONET Path Table function call within the managed object causes the NMS server to generate database access commands to the configuration database within the network device to retrieve data corresponding to each SONET path configured for the selected port. Only some of the data in each row may be necessary to fill in the fields in the tab (e.g., SONET Paths tab 942, Fig. 4w).

Similar to the physical data, logical data is stored in tables within configuration database 42 (Fig. 59). The tables may be provided as part of a default configuration within the configuration database, or the tables may be created within the configuration database as each different type of table is needed. In one embodiment, configuration database 42 includes a SONET Path Table (e.g., 600', Fig. 60g), a Service End Point Table (e.g., 76'', Fig. 60h), an ATM Interface Table (e.g., 114'', Fig. 60i), a Virtual ATM Interface Table (e.g., 993, Fig. 60j), a Virtual Connection Table (e.g., 994, Fig. 60k), a Virtual Link Table (e.g., 995, Fig. 60l) and a Cross-Connect Table (e.g., 996, Fig. 60m). Tables corresponding to other physical layer or upper layer network protocols may also be included within configuration database 42.

The database access commands corresponding to the Get SONET Path Table function call include the port PID (from the proxy / JAVA RMI messages) associated with the selected port. When the database access commands corresponding to the Get SONET Path Table function call are received by the configuration database, the configuration database locates each row in SONET Path Table 600' (Fig. 60g) including the selected port PID and returns to the NMS server the data from each row necessary for the SONET Paths tab. Thus, the retrieved data is limited to those rows / records corresponding to the selected port and the data necessary for the tab. This allows the NMS server and NMS client to quickly respond to the user's request for logical data. If all SONET paths configured for all SONET ports within the network device (or worse, all logical data) were retrieved, then the response time would likely be much slower.

For each row of data the NMS server formats the data according to the SONET Paths tab display and sends it to the NMS client. The NMS client adds the data to the GUI tables which causes the GUI tables to display the SONET paths (e.g., 942a and 942b, Fig. 5h) configured for the selected port. Along with the data necessary for the SONET Paths tab, the NMS server also sends the LID for each logical managed object (i.e., each SONET path) and the NMS client saves the LID within the GUI tables, in one embodiment, within a column hidden from the user.

As previously discussed, to retrieve additional attribute data or change attribute data for a managed object, the user may simply double click the left mouse button on an entry in a tab in configuration / status window 897 (Fig. 5q) to cause a dialog box to appear. When the user double clicks the left mouse button on the entry, the NMS client places a "Get Config" function call to the corresponding proxy and simultaneously opens a GUI dialog 998 (Fig. 59) in local memory 986. If the selected entry is for a physical component of the network device, then the function call causes the NMS client to populate GUI dialog 998 with attribute data from the proxy. If the selected entry is for a logical component of the network device, for example, a SONET path, then the NMS client needs data from the configuration database within the network device to populate GUI dialog 998.

For example, if a user selects SONET path 942a (Fig. 5q) from SONET Paths tab 942 and double clicks the left mouse button, the NMS client displays a SONET Path dialog box 997 (Fig. 62). To do this, when the user double clicks the left mouse button on the entry, the NMS client places a "Get Config" function call to the corresponding port proxy and simultaneously opens a GUI dialog 998 (Fig. 59) in local memory 986. The function call causes the NMS client to send JAVA RMI messages to the NMS server including both the port PID from the proxy and the SONET path LID from the GUI table. The NMS server first searches local memory 987a for the port PID. If a managed object including the port PID is found, then the NMS server issues a "Get Config" function call to the managed object including the SONET Path LID. If the port PID is not found, then the NMS server uses the port PID as a primary key into the configuration database to retrieve data from the row / record corresponding to the port. The NMS server then creates the port managed object, stores it in local memory and issues the "Get Config" function call. The

function call causes the NMS server to generate database access commands and send them to the configuration database within the selected network device.

The database access commands cause the configuration database to retrieve all the attribute data in the row in SONET Path Table 600' (Fig. 60g) corresponding to the SONET path LID. The server uses the retrieved data to build a configuration object and sends the configuration object to the NMS client. The NMS client then uses the configuration object to populate GUI dialog 998 with the data which causes the dialog box 997 (fig. 62) to display the data to the user.

If the user then selects a Cancel button 997a or OK button 997b, then the NMS client closes the dialog box. If the user selects Cancel button 997a, then the NMS client closes and deletes GUI dialog 998 and takes no further action. If the user selects OK button 997b, then it is assumed that the user made changes to one or more SONET path attributes and now wants those changes implemented. To implement any changes made to the SONET path attributes, when the NMS client detects the selection of the OK button, the NMS client places a "Set Config" function call to the corresponding port proxy. The function call causes the NMS client to send JAVA RMI messages to the NMS server including both the port PID from the proxy and the SONET path LID from the GUI table and the attributes for the SONET path. The NMS server first searches local memory 987a for the port PID. If a managed object including the port PID is found, then the NMS server issues a "Set Config" function call to the managed object including the SONET Path LID. If the port PID is not found, then the NMS server uses the port PID as a primary key into the configuration database to retrieve data from the row / record corresponding to the port. The NMS server then creates the port managed object, stores it in local memory and issues the "Set Config" function call. The function call causes the NMS server to generate database access commands and send them to the configuration database within the selected network device.

The database access commands cause the configuration database to locate the row in SONET Path Table 600' (Fig. 60g) corresponding to the SONET path LID and replace the attributes in that row with the attributes included in the database access commands. As discussed in detail above, when tables in the configuration database

are updated an active query feature is used to notify other processes of the changes. For example, NMS database 61 (Fig. 59) is automatically updated with any changes. NMS database 61 may be located within computer/workstation 987 or 984 or within a separate computer / workstation 997. In addition, the changes are sent to the NMS server which uses the data to re-build the configuration object. The NMS server then sends the configuration object to the NMS client. The NMS client uses the configuration object as an indication that the Set Config function call was successful. The NMS client then closes and deletes GUI dialog 998 and uses the received data to update the GUI tables 985.

Alternatively, proxies may be created for each logical managed object and sent to the NMS client. In a typical network device, however, there may be millions of logical managed objects making storage of all logical proxies in memory local to an NMS client difficult if not impossible. Moreover, since logical managed objects change frequently (as opposed to physical managed objects which do not change as frequently), the stored logical proxies would need to be updated frequently leading to an increased burden on both the NMS server and NMS client. Thus, in the preferred embodiment, only physical proxies are created and stored local to the NMS client.

Using the unique PIDs as primary keys allows for faster response times by the NMS server. First the PIDs are used to quickly check local memory 987a – perhaps hash tables in a cache. If the data is not in local memory, the PIDS are used as primary keys to perform a fast data retrieval from configuration database 42. If the PIDs were not used, the NMS server would need to navigate through the hierarchy of tables – possibly performing multiple database accesses -- to locate the data of interest and, thus, response time would be much slower. As primary keys, the PIDs allow the NMS server to directly retrieve required data (i.e., table rows / records) without having to navigate through upper level tables.

Since logical data corresponds to configured objects, rows are added to the tables when logical objects are configured. In addition, the NMS server assigns a unique logical identification number (LID) for each configured object and inserts this within each corresponding row. The LID, like the PID, is used as a primary key within the configuration database for the row / data corresponding to each logical component.

The NMS server and MCD use the same numbering space for LIDs, PIDs and other assigned numbers to ensure that the numbers are different (no collisions). In each row, the NMS server also inserts a unique PID or LID corresponding to a parent table (i.e., a foreign key for association) to provide data "containment".

As described above with reference to Figs. 5a-5p, a user may select a port or a SONET interface and then access a SONET path configuration wizard to configure SONET paths on the selected port / interface. When the user selects OK button 944r, the NMS client places a "Create SONET Path" function call to the proxy corresponding to the selected port / interface including the port PID in the proxy and the parameters provided by the user through the SONET path configuration wizard. The function call causes the NMS client to send JAVA / RMI messages to the NMS server. The NMS server first searches local memory 987a for the port PID. If a managed object including the port PID is found, then the NMS server issues a "Create SONET Path" function call to the managed object including the port PID and the parameters sent by the NMS client. If the port PID is not found, then the NMS server uses the port PID as a primary key into the configuration database to retrieve data corresponding to the port. The NMS server then creates the port managed object, stores it in local memory and then issues the "Create SONET Path" function call. The function call causes the NMS server to generate database access commands and send them to the configuration database within the selected network device.

The database access commands cause the configuration database to add a row in SONET Path Table 600' (Fig. 60g) for each SONET path created by the user. The NMS server assigns a unique path LID 600a (i.e., primary key) to each SONET path and inserts this within the corresponding row. The NMS server also enters data representing attributes for each SONET path (e.g., time slot, number of time slots, etc.) and the unique port PID 600b (i.e., foreign key for association) corresponding to the selected port.

As previously discussed, each SONET path corresponds to a port (e.g., 571a, Fig. 36) on a universal port card (e.g., 554a) and is connected through a cross-connection card (e.g., 562a) to a service end point corresponding to a port (i.e., slice) on a forwarding card (e.g., 546c). In one embodiment, after filling in one or more rows in SONET

Path Table 600', the NMS server also fills in one or more corresponding rows in Service EndPoint Table (SET) 76'' (Fig. 60h). The NMS server assigns a unique service endpoint LID 76a (i.e., primary key) to each service endpoint and inserts the service endpoint LID within a corresponding row. The NMS server also inserts the corresponding path LID 76b (i.e., foreign key for association) within each row and may also insert attributes associated with each service endpoint. For example, the NMS server may insert the quadrant number corresponding to the selected port and may also insert other attributes (if provided by the user) such as the forwarding card slice PID (76d) corresponding to the service end point, the forwarding card PID (76c) on which the port / slice is located and the forwarding card time slot (76e). Alternatively, the NMS server only provides the quadrant number attribute and a policy provisioning manager (PPM) 599 (Fig. 37) decides which forwarding card, slice (i.e., payload extractor chip) and time slot (i.e., port) to assign to the new universal port card path, and once decided, the PPM fills in SET Table 76'' attribute fields (i.e., self-completing configuration record).

For each service endpoint created, the database access commands also cause the configuration database to add a row in an interface table. For example, for each service endpoint corresponding to a SONET path configured for ATM service – that is, service field 942h (Fig. 5q) indicates ATM service -- a row is added to ATM Interface Table 114'' (Fig. 60i). Alternatively, if service field 942h is configured for another service, for example, IP, MPLS or Frame Relay, then a row would be added to an interface table corresponding to that upper layer network protocol. The NMS server assigns a unique ATM interface (IF) LID 114a (i.e., primary key) and within each row inserts both the assigned ATM IF LID 114a and the service endpoint LID 114b (i.e., foreign key for association) corresponding to each ATM interface. The NMS server also inserts in each row data representing attributes (e.g., ATM group number) for each ATM interface. The attribute data may be default values and/or data received within the database access commands.

Again, when tables in the configuration database are updated an active query feature is used to notify other processes including NMS database 61 (Fig. 59) and any NMS server currently connected to the network device, for example, NMS server 851a. Each NMS server builds a configuration object for each changed logical managed

object and sends the configuration object to any NMS clients that currently have access to the network device corresponding to the changed logical managed objects, for example, NMS client 850a. The NMS clients use the received configured object to update GUI tables 985 and display the configuration changes to a user. Thus, the user that created the SONET path(s) would then be able to see the new paths displayed in SONET path tab 942 (Fig. 5q) and new ATM interfaces displayed in ATM interface tab 946 (Fig. 5r).

Similarly, a user may select Virtual ATM Interfaces tab 947 (Fig. 5s) and then select Add button 947b to add a virtual ATM interface to an ATM interface selected in navigation tree 947a. When the user selects OK button 950e (Fig. 5t) in virtual ATM interfaces dialog box 950, the NMS client places an "Add Virtual ATM Interface" function call to the proxy corresponding to the port associated with the selected ATM interface. The function call includes the ATM interface LID (stored in the GUI table), the corresponding port PID and the parameters provided by the user through the ATM interfaces dialog box. The function call causes the NMS client to send JAVA RMI messages to the NMS server. The NMS server first searches local memory 987a for the port PID. If a managed object including the port PID is found, then the NMS server issues an "Add Virtual ATM Interface" function call to the managed object including the ATM interfaces LID and the parameters sent by the NMS client. If the port PID is not found, then the NMS server uses the port PID as a primary key into the configuration database to retrieve data corresponding to the port. The NMS server then creates the port managed object, stores it in local memory and issues the "Add Virtual ATM Interface" function call. The function call causes the NMS server to generate database access commands and send them to the configuration database within the selected network device.

The database access commands cause the configuration database to add a row in Virtual ATM Interfaces Table 993 (Fig. 60d) corresponding to the virtual ATM interface created by the user. The NMS server assigns a unique virtual ATM interface LID 993a (i.e., primary key) to the virtual ATM interface and inserts this within the corresponding row. The NMS server also enters data representing attributes (e.g., A1-An) for the virtual ATM interface and the unique ATM interface LID 993b (i.e., foreign key for association) corresponding to the selected ATM interface in

navigation tree 947a (Fig. 5s). Again, through the active query feature, the NMS database and NMS server are notified of the changes made to the configuration database. The NMS server builds a configuration object and sends it to the NMS client which updates the GUI tables to display the added virtual ATM interface (e.g., 947c, Fig. 5u) to Virtual ATM Interfaces tab 947. The configuration object may be temporarily stored in local memory 986. However, once the GUI tables are updated, the NMS client deletes the configured object from local memory 986.

Because there may be many upper layer network protocol interfaces in network device 540, the port managed object and port proxy may become very large as more and more function calls (e.g., Add Virtual ATM Interface, Add Virtual MPLS Interface, etc.) are added for each type of interface. To limit the size of the port managed object and port proxy, all interface function calls may be added to logical proxies corresponding to logical upper layer protocol nodes. For example, an ATM node table 999 (Fig. 60n) may be included in configuration database 42, and when ATM service is first configured by a user on network device 540, the NMS server assigns an ATM node LID 999a (e.g., 5000) and inserts the ATM node LID and the managed device PID 999b (e.g., 1) in one row 999c in the ATM node table. The NMS server may also insert any attributes (A1-An). The NMS server then retrieves the data in the row and creates an ATM logical managed object (ATM LMO). Like the physical managed objects, the ATM logical managed object includes the assigned LID (e.g., 5000), attribute data and function calls. The function calls include Get Proxy and interface related function calls like Add Virtual ATM Interface. The NMS server stores the ATM LMO in local memory 987a and issues a Get Proxy function call. After creating the ATM proxy (ATM PX), the NMS server sends the ATM proxy to memory 986 local to NMS client 850a. The NMS client uses the ATM proxy to update GUI tables 985, and then uses it to later make function calls to get ATM interface related data from configuration database 42.

Thus, after the user selects OK button 950e (Fig. 5t) in virtual ATM interfaces dialog box 950, the NMS client places an "Add Virtual ATM Interface" function call to the ATM node proxy. The function call includes the ATM interface LID (stored in the GUI table), the corresponding ATM node LID and the parameters provided by the user through the ATM interfaces dialog box. The function call causes the NMS client

to send JAVA RMI messages to the NMS server. The NMS server first searches local memory 987a for the ATM node LID. If a managed object including the ATM node LID is found, then the NMS server issues an "Add Virtual ATM Interface ATM interface LID and the parameters sent by the NMS client. If the ATM node LID is not found, then the NMS server uses the ATM node LID as a primary key into the configuration database to retrieve data corresponding to the port. The NMS server then creates the ATM node logical managed object, stores it in local memory and issues the "Add Virtual ATM Interface" function call. The function call causes the NMS server to generate database access commands and send them to the configuration database within the selected network device.

The database access commands cause the configuration database to add a row in Virtual ATM Interfaces Table 993 (Fig. 60d) corresponding to the virtual ATM interface created by the user. The NMS server assigns a unique virtual ATM interface LID 993a (i.e., primary key) to the virtual ATM interface and inserts this within the corresponding row. The NMS server also enters data representing attributes (e.g., A1-An) for the virtual ATM interface and the unique ATM interface LID 993b (i.e., foreign key for association) corresponding to the selected ATM interface in navigation tree 947a (Fig. 5s). Again, through the active query feature, the NMS database and NMS server are notified of the changes made to the configuration database. The NMS server builds a configuration object and sends it to the NMS client which updates the GUI tables to display the added virtual ATM interface (e.g., 947c, Fig. 5u) to Virtual ATM Interfaces tab 947. The NMS client then deletes the logical managed objects from local memory 986." function call to the managed object including the

In the discussion below, virtual connections are added using the ATM node proxy. It should be understood, however, that a port proxy including the virtual connection function calls could be used instead.

As explained above, to add a virtual connection, the user may select a port (e.g., 941a, Fig. 5v) and then select the "Add Virtual Connection" option from pull down menu 943 or the user may select a virtual ATM interface (e.g., 947c, Fig. 5v) in Virtual ATM Interfaces tab 947 and then select Virtual Connections button 947d. After

creating a virtual connection through Virtual Connection Wizard 952 (Figs. 5w-5x), the user selects Finish button 953w. This causes the NMS client to place an "Add Virtual Connection" function call to the ATM node proxy. The function call includes the virtual ATM interface LID (stored in the GUI table), the corresponding ATM node PID and the parameters provided by the user through the Virtual Connection Wizard. The function call causes the NMS client to send JAVA RMI messages to the NMS server. The NMS server first searches local memory 987a for the ATM node LID. If a managed object including the ATM node LID is found, then the NMS server issues an "Add Virtual Connection" function call to the managed object including the virtual ATM interface LID and the parameters sent by the NMS client. If the ATM node LID is not found, then the NMS server uses the ATM node LID as a primary key into the configuration database to retrieve data corresponding to the ATM node. The NMS server then creates the ATM node logical managed object, stores it in local memory and then issues the "Add Virtual Connection" function call. The function call causes the NMS server to generate database access commands and send them to the configuration database within the selected network device.

The database access commands cause the configuration database to add a row in Virtual Connection Table 994 (Fig. 60k) corresponding to the virtual connection created by the user. The NMS server assigns a unique virtual connection LID 994a (i.e., primary key) to the virtual connection and inserts this within the corresponding row. The NMS server also enters data representing attributes (e.g., A1-An) for the virtual connection and the unique virtual ATM interface LID 994b (i.e., foreign key for association) corresponding to the selected virtual ATM interface in Virtual ATM Interfaces tab 947 (Fig. 5v).

In addition to adding a row to Virtual Connection table 994, when a virtual connection is created one or more rows are also added to Virtual Link Table 995 (Fig. 60l) and Cross-Connection Table 996 (Fig. 60m). With regard to Virtual Link Table 995, the NMS server assigns a unique virtual link LID 995a (i.e., primary key) to each endpoint in the virtual connection and inserts each endpoint LID within a row in the Virtual Link Table. The NMS server also enters data in each row representing attributes (e.g., A1-An) for the corresponding endpoint and the unique virtual connection LID 995b (i.e., foreign key for association) corresponding to the newly

created virtual connection 994a (Fig. 60k). For a point-to-point connection there will be two end points – that is, two rows are added to the Virtual Link Table each including a unique endpoint LID 995a and the same virtual connection LID 995b (corresponding to the same virtual connection LID 994a, Fig. 60k). For a point to multipoint connection there will be one source endpoint and multiple destination endpoints – that is, more than two rows are added to the Virtual Link Table, one row corresponding to the source endpoint and one row corresponding to each destination endpoint, where each row includes a unique endpoint LID 995a and the same virtual connection LID 995b (corresponding to the same virtual connection LID 994a, Fig. 60k).

Each row / record in Cross-Connection Table 60g, represents the relationship between the various endpoints and virtual connections. One row is created for each point-to-point connection while multiple rows are created for each point-to-multipoint connection. The NMS server assigns a unique cross-connection LID 996a (i.e., primary key) to each cross-connection and inserts each cross-connection LID within a row in the Cross-Connection Table. The NMS server also enters data in each row representing attributes (e.g., A1-An) for the corresponding cross-connection. The NMS server then enters two foreign keys for association: Virtual Link 1 LID 996b and Virtual Link 2 LID 996c. Within Virtual Link 1 LID 996b the NMS server inserts the source endpoint LID for the virtual connection. Within Virtual Link 2 LID 996c, the NMS server inserts a destination endpoint LID for the virtual connection. For each of these Virtual Link LIDs in Virtual Link Table 995, the NMS server also inserts Cross-Connection LID 995c (corresponding to Cross-Connection LID 996a in Cross-Connection Table 996). Since a point-to-point connection includes only one destination endpoint, only one row in the Cross-Connection table is needed to fully represent the connection. One or more rows are necessary, however, to represent a point-to-multipoint connection. In each of the other rows, Virtual Link 1 LID 996b representing the source endpoint remains the same but in each row a different Virtual Link 2 LID 996c is added representing the various destination endpoints.

Again, through the active query feature, the NMS database and NMS server are notified of the changes made to the Virtual Connection Table, Virtual Link Table and Cross-Connection Table in the configuration database. The NMS server creates

configuration objects for each changed row and sends the configuration objects to the NMS client which updates the GUI tables to display the added virtual connection (e.g., 948a, Fig. 5z) in the Virtual Connections tab 948.

In addition to adding rows to tables when logical managed objects are configured, rows are also removed from tables when logical managed objects are deleted. For example, if a user selects a SONET path (e.g., 942a, Fig. 5q) from SONET Paths Tab 942 and then selects Delete button 942g, the NMS client places a "Delete SONET Path" function call to the proxy corresponding to the selected port. The function call includes the selected port PID as well as the SONET Path LID corresponding to the SONET path to be deleted. The function call causes the NMS client to send JAVA RMI messages to the NMS server. The NMS server first searches local memory 987a for the port PID. If a managed object including the port PID is found, then the NMS server issues a "Delete SONET Path" function call to the managed object including the SONET path LID. If the port PID is not found, then the NMS server uses the port PID as a primary key into the configuration database to retrieve data from the row / record corresponding to the port. The NMS server then creates the port managed object, stores it in local memory and issues the "Delete SONET Path" function call. The function call causes the NMS server to generate database access commands and send them to the configuration database within the selected network device.

The database access commands cause the configuration database to directly delete the specific row within SONET Path Table 600' (Fig. 60g) corresponding to the SONET path LID (primary key). Through the active query feature, the NMS database and NMS server are notified of the changes made to the SONET Path Table in the configuration database. The NMS server sends JAVA RMI messages to the NMS client to cause the client to update the GUI tables to remove the deleted SONET Path from the SONET Paths tab 942.

Many different function calls may be generated by the NMS client and NMS server to carry out configuration changes requested by users.

As described above, memory local to each NMS client is utilized to store proxies corresponding to managed objects associated with physical components within a

network device selected by a user. Proxies for logical managed objects corresponding to upper layer network protocol nodes (e.g., ATM node, IP node, MPLS node, Frame Relay node, etc.) may also be stored in memory local to each NMS client to limit the size of physical port proxies. The proxies reduce the load on the network / NMS server by allowing the NMS client to respond to user requests for physical network device data and views without having to access the NMS server. Storing data local to the NMS client improves the scalability of the NMS server by not requiring the NMS server to maintain the managed objects in memory local to the server. Thus, as multiple NMS clients request access to different network devices, the NMS server may, if necessary, overwrite managed objects within its local memory without disrupting the NMS client's ability to display physical network device information to the user and issue function calls to the NMS server. Response time to a user's request for access to a network device is also improved by initially only retrieving physical data as opposed to retrieving both physical and logical data.

In addition, unique identification numbers – both PIDs and LIDs – may also be stored in memory local to the NMS client (e.g., within proxies or GUI tables) to provide improved data request response times. Instead of navigating through the hierarchy of tables within the relational configuration database internal to the network device, the NMS server is able to use the unique identification numbers as primary keys to directly retrieve the specific data needed. Providing the unique identification numbers from the NMS client to the NMS server insures that even if the NMS server needed to overwrite managed objects within memory local to the NMS server, the NMS server will be able to quickly re-generate the managed objects and quickly retrieve the necessary data.

The unique identification numbers – both PIDs and LIDs – may be used in a variety of ways. For example, as previously mentioned, the device mimic 896a (Fig. 4t) is linked with status window 897, such that selecting a module in device mimic 896a causes the Module tab to highlight a line in the inventory corresponding to that card. The unique PIDs and LIDs are utilized to make this link between the status window and the device mimic.

It will be understood that variations and modifications of the above described methods and apparatuses will be apparent to those of ordinary skill in the art and may be made without departing from the inventive concepts described herein. Accordingly, the embodiments described herein are to be viewed merely as illustrative, and not limiting, and the inventions are to be limited solely by the scope and spirit of the appended claims.

Docket No. 50277-2029

Claims:

1. A method of managing a telecommunications network, comprising:
storing user profile data corresponding to a user profile in a first data repository;
storing network device data corresponding to a network device in the telecommunications network in a second data repository;
detecting a request from a user for network device data corresponding to the network device, wherein the user request is associated with the user profile;
generating a data access request to the second data repository utilizing the user profile data from the first data repository; and
retrieving network device data from the second data repository in accordance with the user request.

2. The method of claim 1, wherein the first data repository is a central data repository.

3. The method of claim 1, wherein the second data repository is embedded within the network device.

4. The method of claim 1, wherein the first and second data repositories are databases.

5. The method of claim 1, further comprising:
displaying the retrieved network device data in a user interface.

6. The method of claim 1, wherein the user profile data includes an IP address assigned to the network device.

7. The method of claim 6, wherein the user profile data further includes a port identification for a port on the network device.

8. The method of claim 1, wherein the user profile data includes a Domain Name Server name assigned to the network device.

9. The method of claim 1, wherein the user profile data includes a group access level.

10. The method of claim 9, wherein the user profile data further includes a password corresponding to the group access level for gaining access to the network device.

11. The method of claim 1, wherein the user profile data includes an simple network management protocol (SNMP) community string.

12. The method of claim 1, wherein the user profile data includes one or more group names corresponding to the network device and wherein generating a data access request to the second data repository using user profile data from the first data repository comprises:

generating a data access request to the second data repository for each group name in the user profile data; and

wherein retrieving network device data from the second data repository in accordance with the user request, comprises, for each data access request:

searching the second data repository for the group name corresponding to the data access request; and

retrieving network device data from the second data repository corresponding to the group name if the group name corresponding to the data access request is found in the second data repository.

13. The method of claim 12, wherein the network device data retrieved from the second data repository comprises configured resource data.

14. The method of claim 12, wherein retrieving network device data from the second data repository corresponding to the group name, comprises:

using the group name to dynamically determine which data in the second data repository is retrieved.

15. The method of claim 12, wherein the first and second data repositories are databases and wherein retrieving network device data from the second database corresponding to the group name, comprises:

using the group name in a database query to actively filter which data in the second database is retrieved.

16. The method of claim 15, wherein generating a data access request to the second database for each group name in the user profile data from the first database comprises:

generating a where clause including the group name; and

sending the where clause to the second database.

17. The method of claim 1, wherein the first and second data repositories are relational databases and the user profile data is stored in at least one table within the first database and network device data is stored in at least one table within the second database.

18. The method of claim 1, further comprising:

generating a user profile logical managed object (LMO) including at least a portion of the user profile data from the first data repository; and

wherein generating a data access request to the second data repository utilizing the user profile data from the first data repository, comprises:

generating a data access request to the second data repository utilizing the user profile data from the user profile LMO.

19. The method of claim 18, wherein prior to generating a user profile LMO, the method further comprises:

detecting a user log-on request.

20. The method of claim 1, wherein detecting a request from a user for network device data, comprises:

detecting the user request through a network management system (NMS) client; and

sending the user request from the NMS client to an NMS server;

wherein the NMS server generates the data access request to the second data repository utilizing the user profile data from the first data repository and retrieves network device data from the second data repository in accordance with the user request; and

wherein the method further comprises:

sending the retrieved network device data from the NMS server to the NMS client.

21. The method of claim 20, further comprising:

generating a user profile logical managed object (LMO) at the NMS server, wherein the user profile LMO includes at least a portion of the user profile data from the first data repository;

sending the NMS client the user profile LMO; and

wherein generating a data access request to the second data repository utilizing the user profile data from the first data repository, comprises:

generating a data access request to the second data repository utilizing the user profile data from the user profile LMO.

22. The method of claim 20, further comprising:

generating a user profile LMO at the NMS server, wherein the user profile LMO includes at least a portion of the user profile data from the first data repository;

generating a client user profile LMO at the NMS server, wherein the client user profile LMO includes at least a portion of the user profile data from the first data repository in a format expected by the NMS client;

sending the client user profile LMO to the NMS client; and

wherein generating a data access request to the second data repository utilizing the user profile data from the first data repository, comprises:

generating a data access request to the second data repository utilizing the user profile data from the client user profile LMO.

23. The method of claim 1, wherein the user request is a first user request, the network device is a first network device and the data access request is a first data access request, and wherein the method further comprises:

storing network device data corresponding to a second network device in the telecommunications network in a third data repository;

detecting a second user request from the user for network device data corresponding to the second network device, wherein the second user request is associated with the user profile;

generating a second data access request to the third data repository utilizing the user profile data from the first data repository; and

retrieving network device data from the third data repository in accordance with the second user request.

24. The method of claim 1, wherein the user profile data is first user profile data, the user is a first user, the user request is a first user request and the data access request is a first data access request, and wherein the method further comprises:

storing second user profile data corresponding to a second user profile in the first data repository;

detecting a second request from a second user for network device data corresponding to the network device, wherein the second user request is associated with the second user profile;

generating a second data access request to the second data repository utilizing the second user profile data from the first data repository; and

retrieving network device data from the second data repository in accordance with the second user request.

25. The method of claim 1, wherein the user profile data is first user profile data, the user is a first user, the user request is a first user request, the network device is a first network device and the data access request is a first data access request, and wherein the method further comprises:

storing second user profile data corresponding to a second user profile in the first data repository;

storing network device data corresponding to a second network device in the telecommunications network in a third data repository;

detecting a second request from a second user for network device data corresponding to the second network device, wherein the second user request is associated with the second user profile;

generating a second data access request to the third data repository utilizing the second user profile data from the first data repository; and

retrieving network device data from the third data repository in accordance with the second user request.

26. A method of managing a telecommunications network, comprising:

storing user profile data corresponding to a user profile in a first data repository, wherein the user profile data includes a group name;

storing network device data corresponding to a network device in the telecommunications network in a second data repository;

detecting a request from a user for network device data corresponding to the network device, wherein the user request is associated with the user profile;

generating a data access request to the second data repository utilizing the user profile data from the first data repository, wherein the data access request includes the group name; and

retrieving network device data from the second data repository in accordance with the group name.

27. The method of claim 26, wherein the first data repository is a central data repository.

28. The method of claim 26, wherein the second data repository is embedded within the network device.

29. The method of claim 26, wherein the first and second data repositories are databases.

30. The method of claim 26, wherein retrieving network device data from the second data repository in accordance with the group name, comprises:

searching the second data repository for the group name; and

retrieving network device data from the second data repository corresponding to the group name if the group name is found in the second data repository.

31. The method of claim 26, wherein the network device data retrieved from the second data repository comprises configured resource data associated with the group name.

32. The method of claim 26, wherein retrieving network device data from the second data repository corresponding to the group name, comprises:

using the group name to dynamically determine which data in the second data repository is retrieved.

33. The method of claim 26, wherein the first and second data repositories are databases and wherein retrieving network device data from the second database corresponding to the group name, comprises:

using the group name in a database query to actively filter which data in the second database is retrieved.

34. The method of claim 33, wherein generating a data access request to the second database comprises:

generating a where clause including the group name; and

sending the where clause to the second database.

35. The method of claim 26, wherein the user profile data includes a plurality of group names and wherein generating a data access request to the second data repository utilizing the user profile data from the first data repository, comprises:

generating a data access request to the second data repository for each group name in the user profile data, wherein each data access request includes the corresponding group name; and

wherein retrieving network device data from the second data repository in accordance with the group name, comprises:

retrieving network device data from the second data repository in accordance with each group name.

36. A method of managing a telecommunications network, comprising:

storing user profile data corresponding to a user profile in a central data repository;

detecting a user log-on request corresponding to the user profile through a network management system (NMS); and

generating a user profile logical managed object (LMO) using the user profile data stored in the central data repository.

37. The method of claim 36, further comprising:

storing network device data in a network device data repository embedded within a network device in the telecommunications network, wherein the data corresponds to configured resources within the network device;

detecting through the NMS a request for data corresponding to the configured resources within the network device from a user associated with the user profile; and

retrieving network device data from the network device data repository in accordance with the user profile data and the request for data.

38. The method of claim 37, wherein the user profile data includes one or more group names, wherein the user profile LMO includes the one or more group names, and wherein retrieving network device data from the network device data repository in accordance with the user profile data and the request for data, comprises:

searching the network device data repository for a match with each group name in the user profile LMO;

returning an empty data set for each group name not found in the network device data repository; and

retrieving data corresponding to configured resources from the network device data repository for each group name found in the network device data repository.

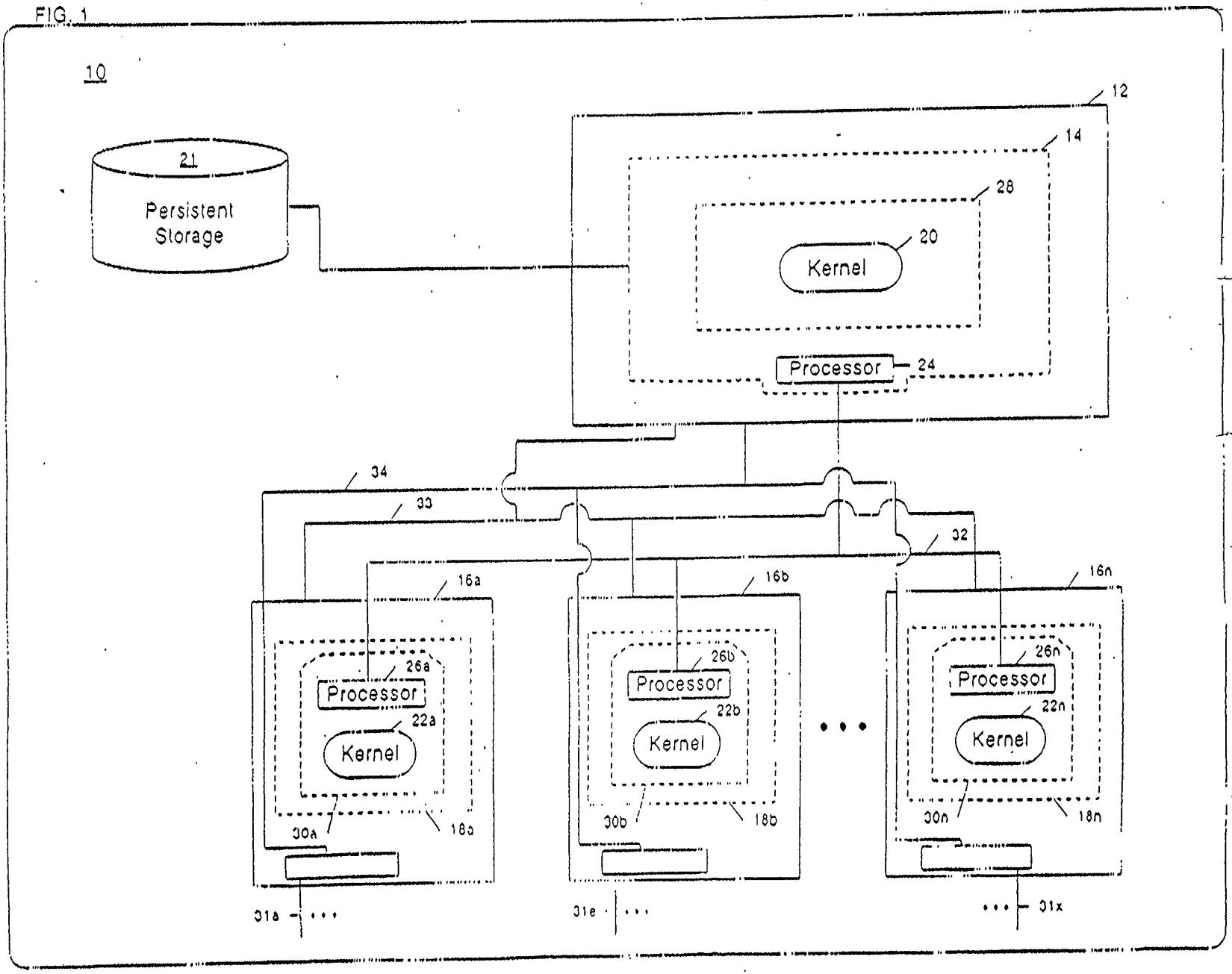
39. The method of claim 37, wherein the central data repository and network device data repository comprise databases.

Abstract of the Disclosure

The present invention provides a method and apparatus for accessing network device data through user profiles. User profiles may be created by network administrators, and the corresponding user profile data may be stored in a central network management system (NMS) database. When a user requests data from a particular network device, the NMS utilizes the user profile data in the central database to access the network device and retrieve the required network device data. Since the user profile data is stored in a central database, the user may log into the NMS from any location and connect to any network device in the network. Further, the user profile data may be used to limit which network devices and which network device configured resources the user may access as well as the user's access level.

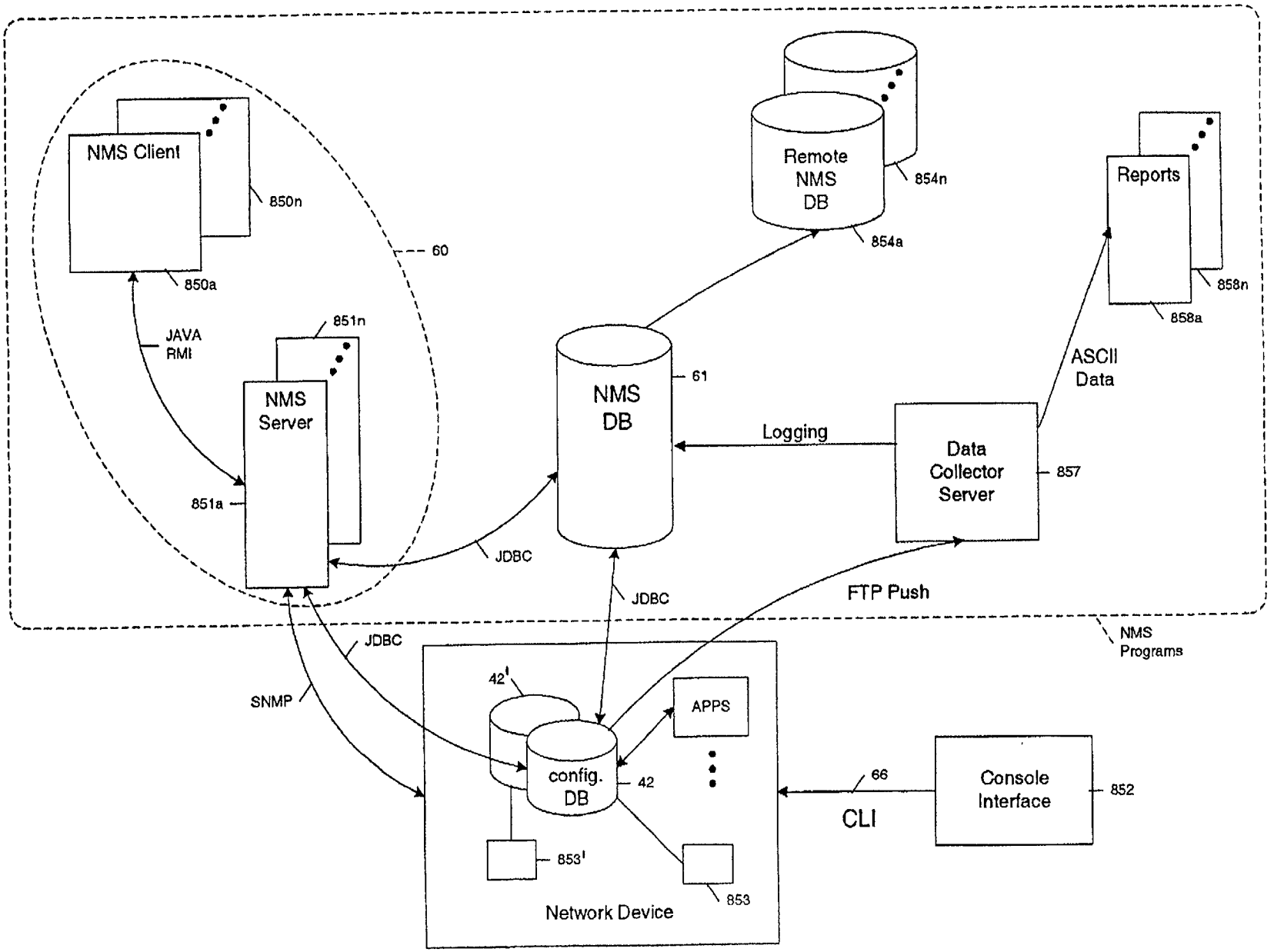
DRAFT DESIGNED

FIG. 1



Sent By: Equipe Communications; 781 795 2031; Oct-9-00 6:52PM; Page 2

Fig. 2a



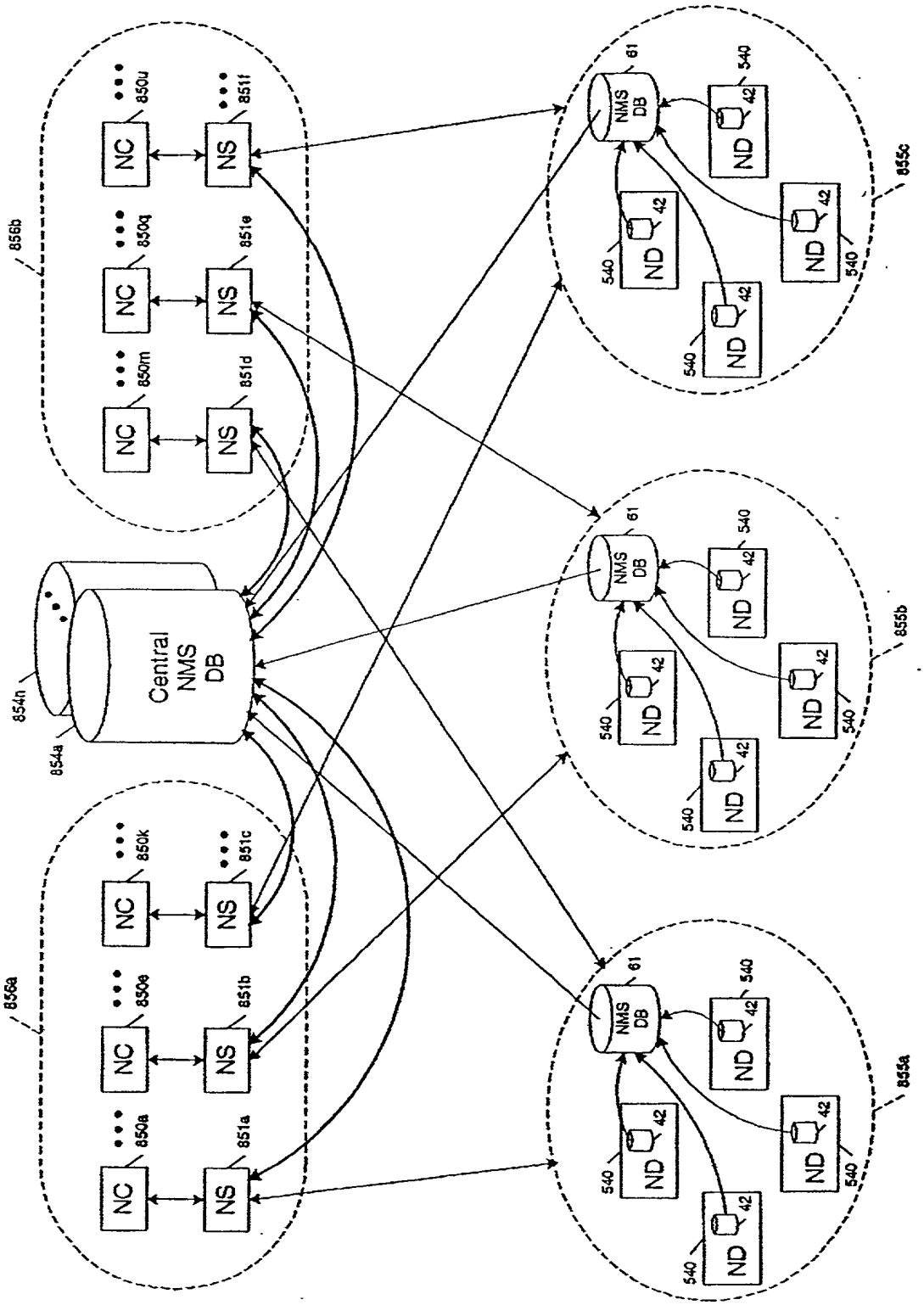

Central NMS DB
854n
854a
856a
856b
855a
855b
855c
NC
NS
ND
NMS DB
850a
850e
850k
850m
850q
850u
851a
851b
851c
851d
851e
851f
540
42
61

Fig. 2b

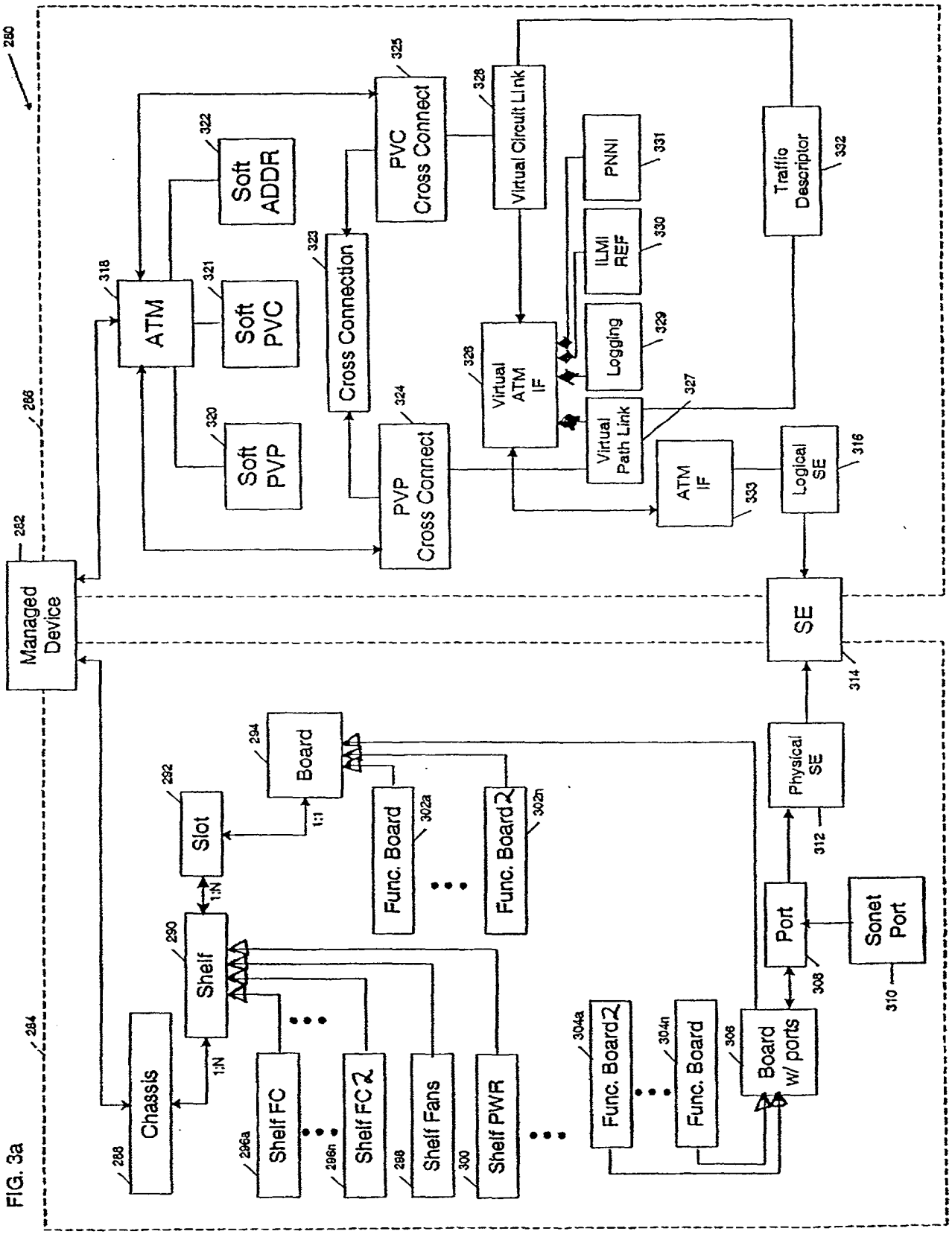

280
286
282
Managed Device
284
Chassis
288
Shelf
290
1:N
1:N
Slot
292
1:1
Board
294
Shelf FC
296a
Shelf FC2
296n
Shelf Fans
298
Shelf PWR
300
Func. Board
302a
Func. Board2
302n
Func. Board2
304a
Func. Board
304n
Board w/ ports
306
Port
308
Sonet Port
310
Physical SE
312
SE
314
Logical SE
316
ATM
318
Soft PVP
320
Soft PVC
321
Soft ADDR
322
Cross Connection
323
PVP Cross Connect
324
PVC Cross Connect
325
ATM IF
333
Virtual Path Link
327
Virtual ATM IF
326
Virtual Circuit Link
328
Logging
329
ILMI REF
330
PNNI
331
Traffic Descriptor
332

FIG. 3a

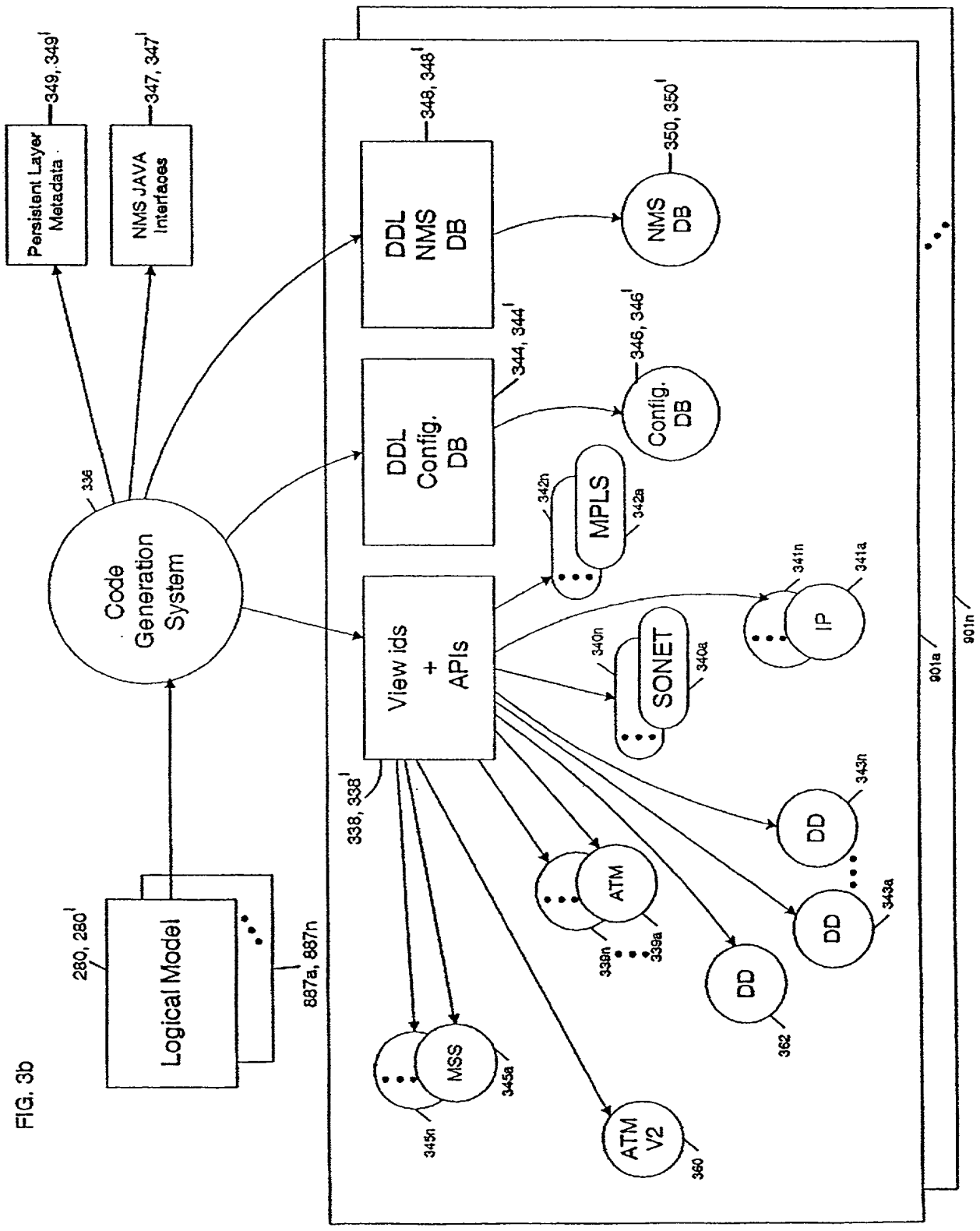

Persistent Layer Metadata
349, 349'
NMS JAVA Interfaces
347, 347'
336
Code Generation System
Logical Model
280, 280'
887a, 887n
DDL NMS DB
348, 348'
NMS DB
350, 350'
DDL Config. DB
344, 344'
Config. DB
346, 346'
View ids + APIs
338, 338'
MPLS
342n
342a
SONET
340n
340a
IP
341n
341a
ATM
338n
339a
DD
343n
DD
343a
DD
362
MSS
345n
345a
ATM V2
360
901a
901n

FIG. 3b

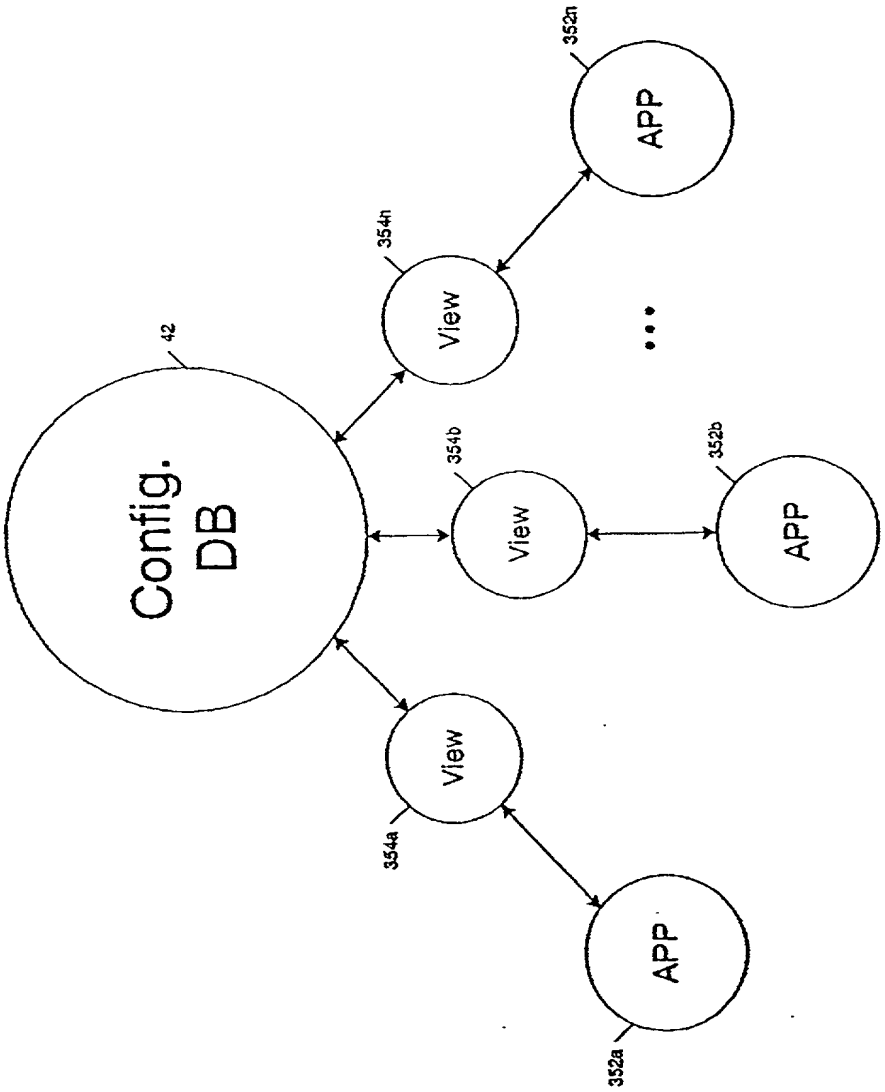
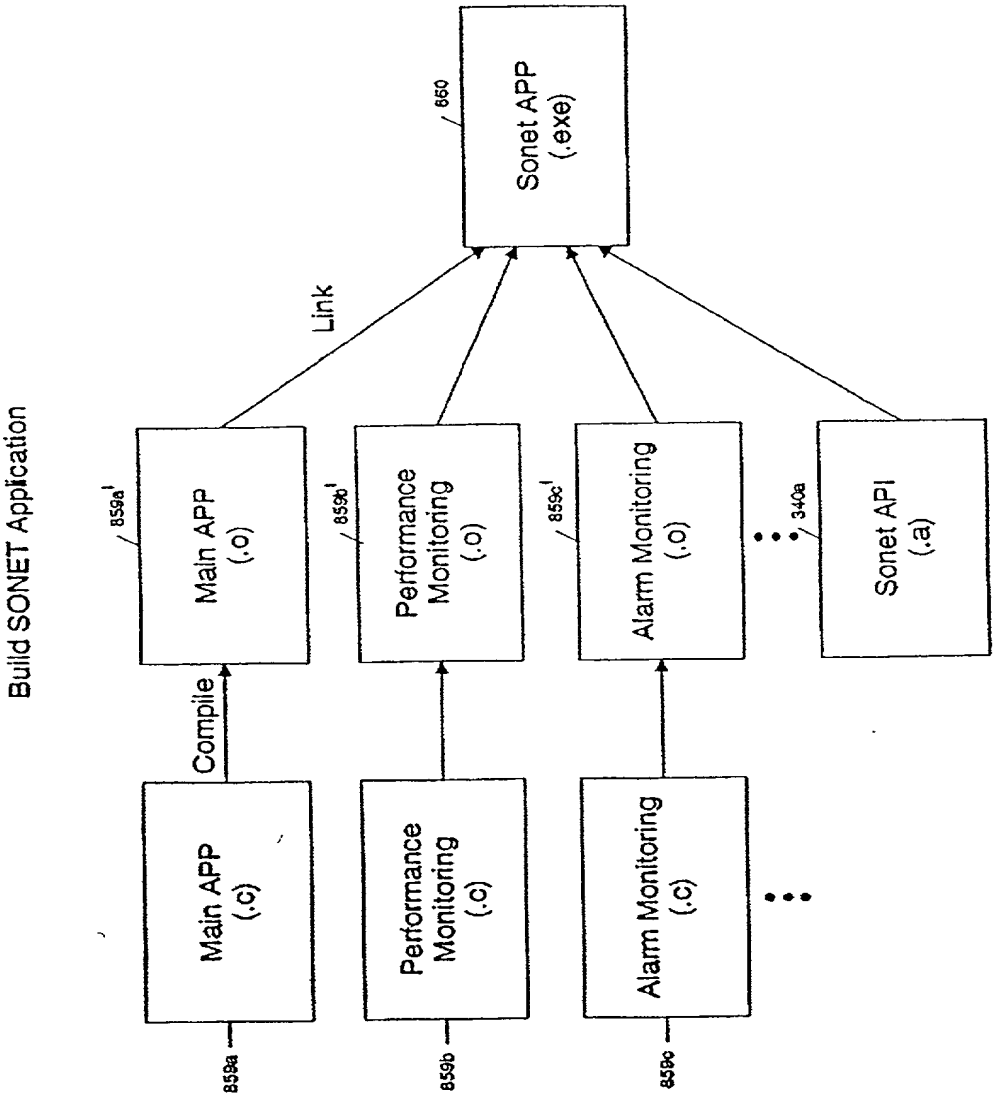

Config. DB
42
View
354a
View
354b
View
354n
APP
352a
APP
352b
APP
352n

FIG. 3c

Fig. 3d


Build SONET Application
Main APP (.c)
Performance Monitoring (.c)
Alarm Monitoring (.c)
Compile
Main APP (.o)
Performance Monitoring (.o)
Alarm Monitoring (.o)
Sonet API (.a)
Link
Sonet APP (.exe)
858a
858b
858c
859a'
859b'
859c'
340a
860

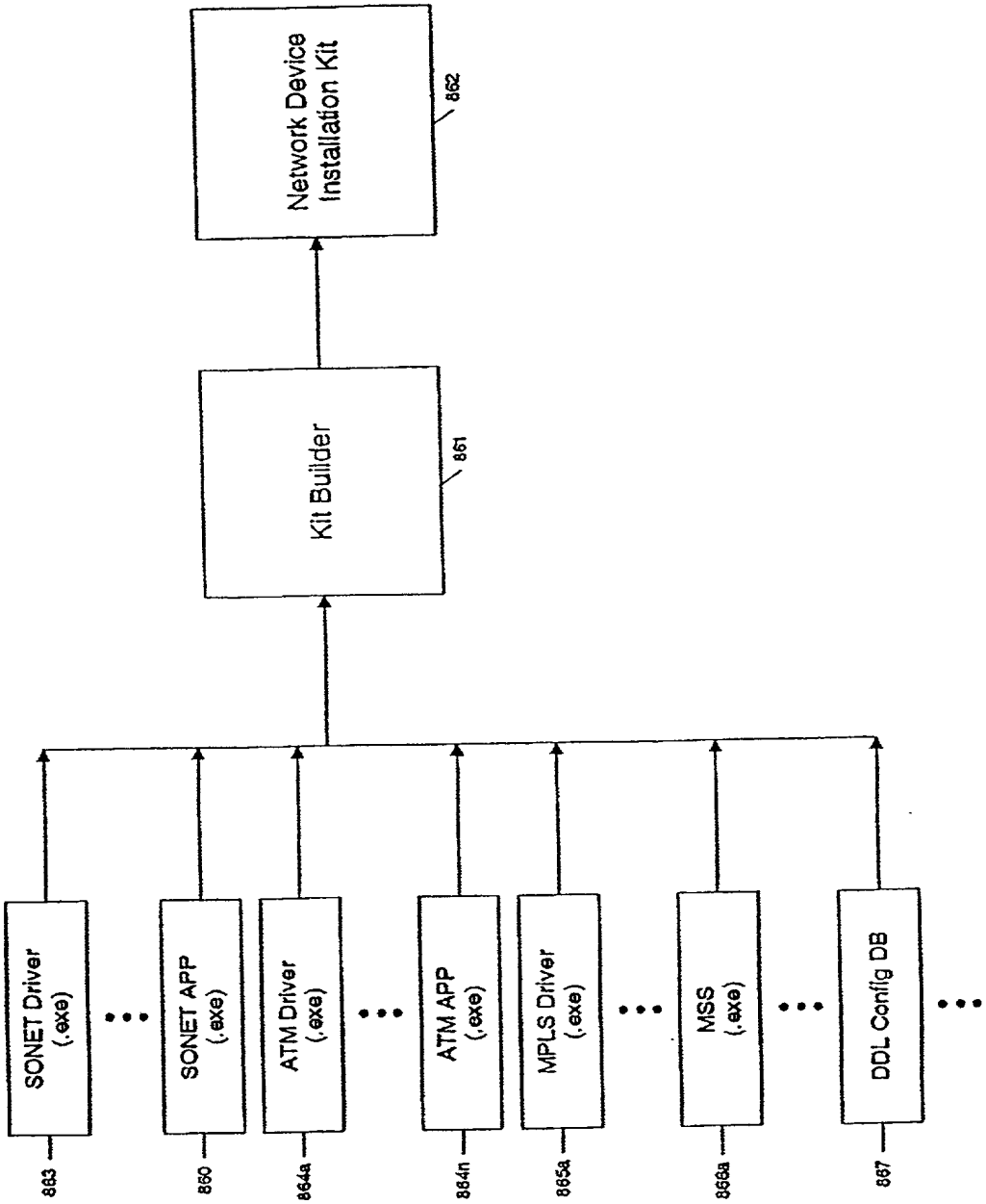
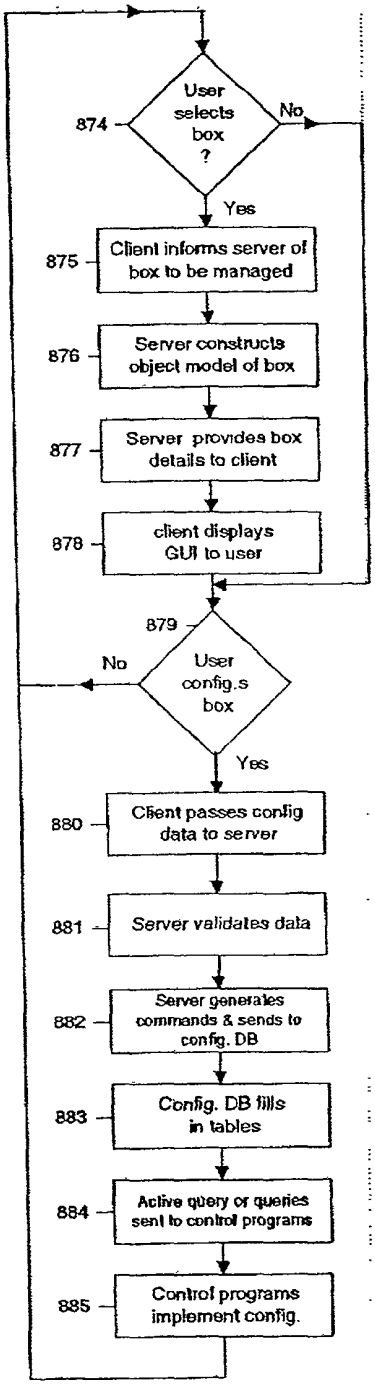

Network Device Installation Kit
862
Kit Builder
861
SONET Driver (.exe)
863
SONET APP (.exe)
860
ATM Driver (.exe)
864a
ATM APP (.exe)
864n
MPLS Driver (.exe)
865a
MSS (.exe)
866a
DDL Config DB
867

Fig. 3e

Fig. 3B

NMS Installation Kit
871
Kit Builder
861
DDL NMS DB
868
NMS JAVA Interfaces
869
Persistent Layer Metadata
870
NMS Server
865
NMS Client
866

Fig. 3g


874
User selects box ?
No
Yes
875
Client informs server of box to be managed
876
Server constructs object model of box
877
Server provides box details to client
878
client displays GUI to user
879
User config.s box
No
Yes
880
Client passes config data to server
881
Server validates data
882
Server generates commands & sends to config. DB
883
Config. DB fills in tables
884
Active query or queries sent to control programs
885
Control programs implement config.

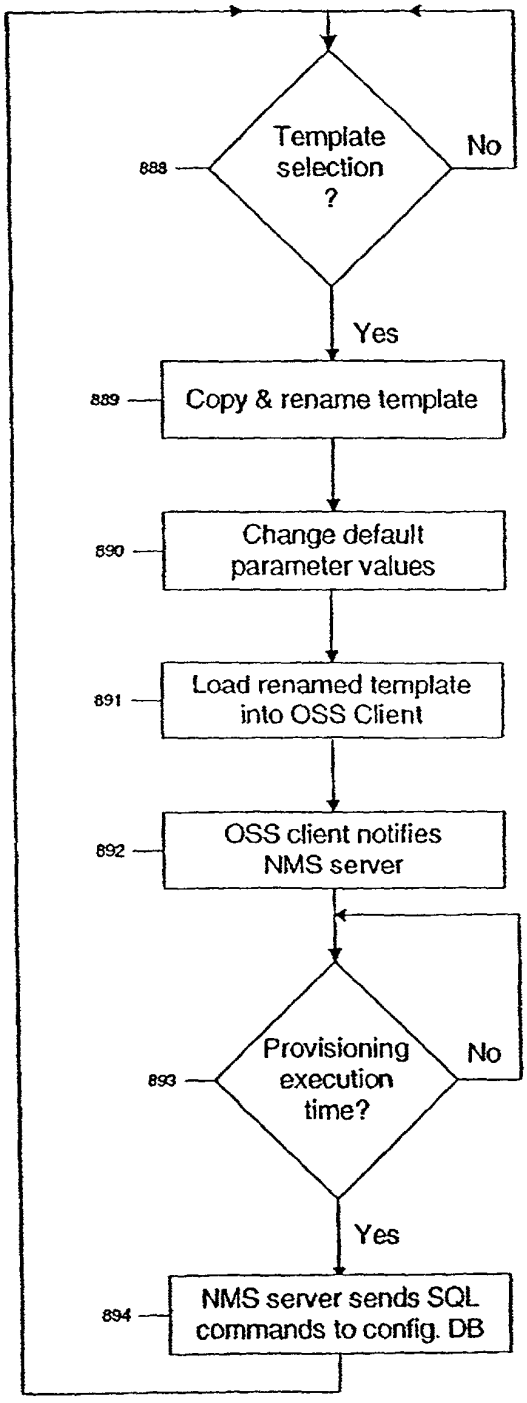


Fig. 3h

09/24/2003 09:56:29

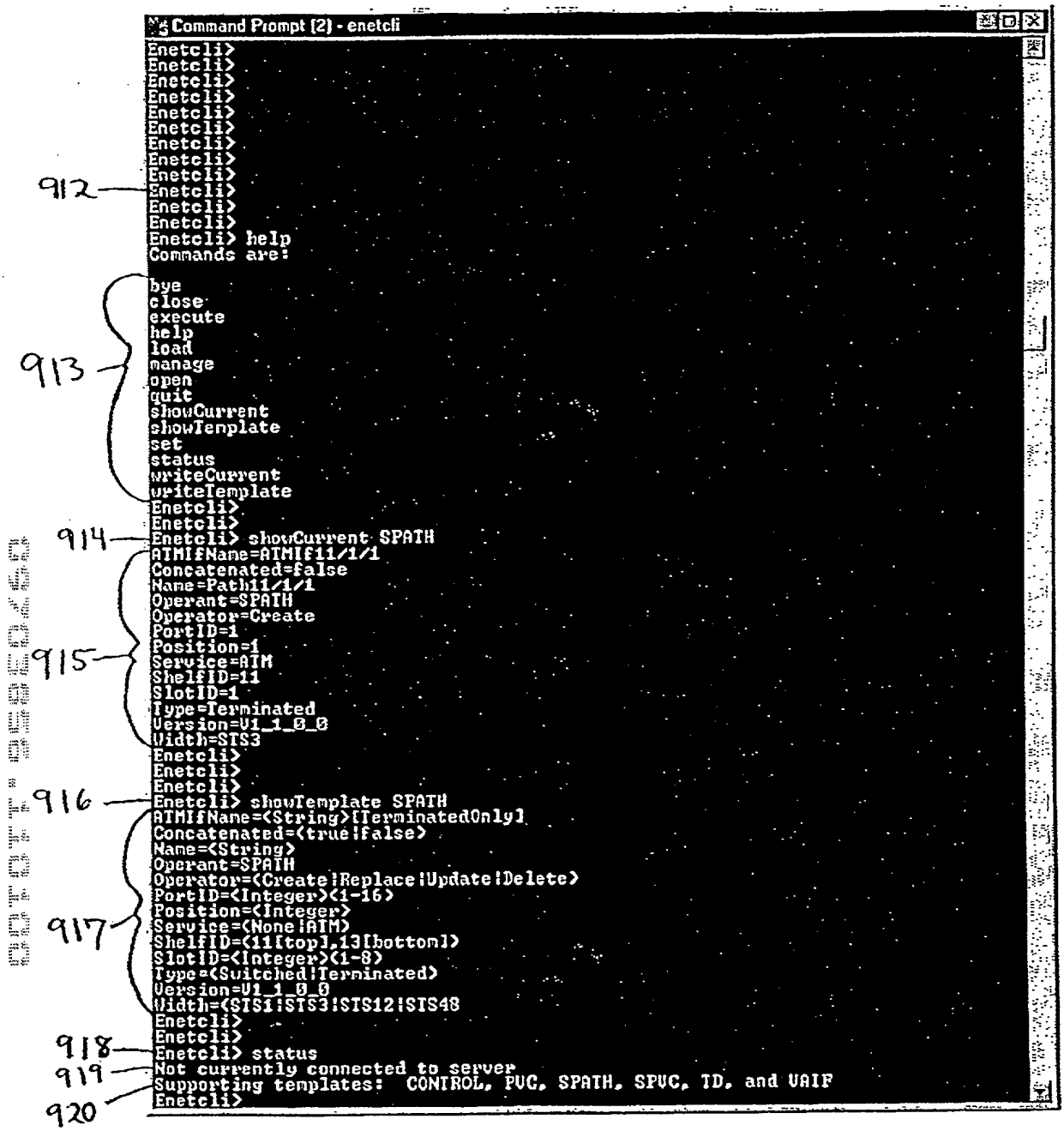

Command Prompt (2) - enetcli
Enetcli>
Enetcli> help
Commands are:
bye
close
execute
help
load
manage
open
quit
showCurrent
showTemplate
set
status
writeCurrent
writeTemplate
Enetcli> showCurrent SPATH
ATMIfName=ATMIf11/1/1
Concatenated=false
Name=Path11/1/1
Operant=SPATH
Operator=Create
PortID=1
Position=1
Service=AIM
ShelfID=11
SlotID=1
Type=Terminated
Version=V1_1_0_0
Width=STS3
Enetcli> showTemplate SPATH
ATMIfName=<String>[TerminatedOnly]
Concatenated=<true|false>
Name=<String>
Operant=SPATH
Operator=<Create|Replace|Update|Delete>
PortID=<Integer>(1-16)
Position=<Integer>
Service=<None|AIM>
ShelfID=<11[top],13[bottom]>
SlotID=<Integer>(1-8)
Type=<Switched|Terminated>
Version=V1_1_0_0
Width=<STS1|STS3|STS12|STS48
Enetcli> status
Not currently connected to server
Supporting templates: CONTROL, PVC, SPATH, SPVC, TD, and VAIF
Enetcli>
912
913
914
915
916
917
918
919
920

Fig. 3i

Fig. 3j

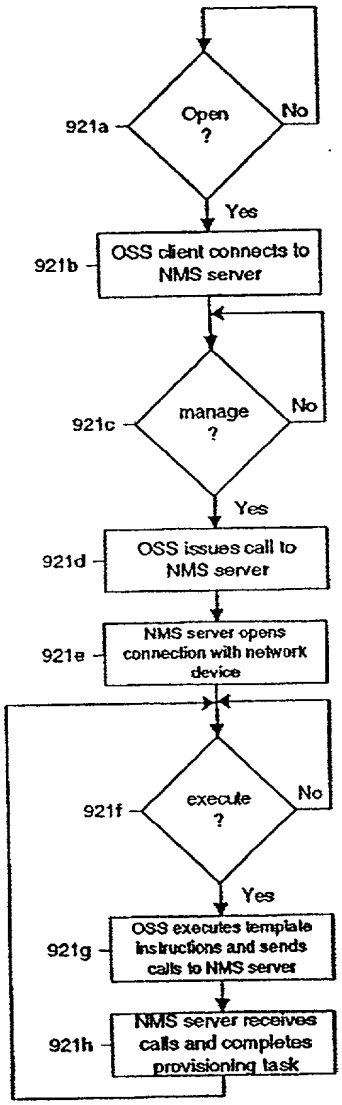
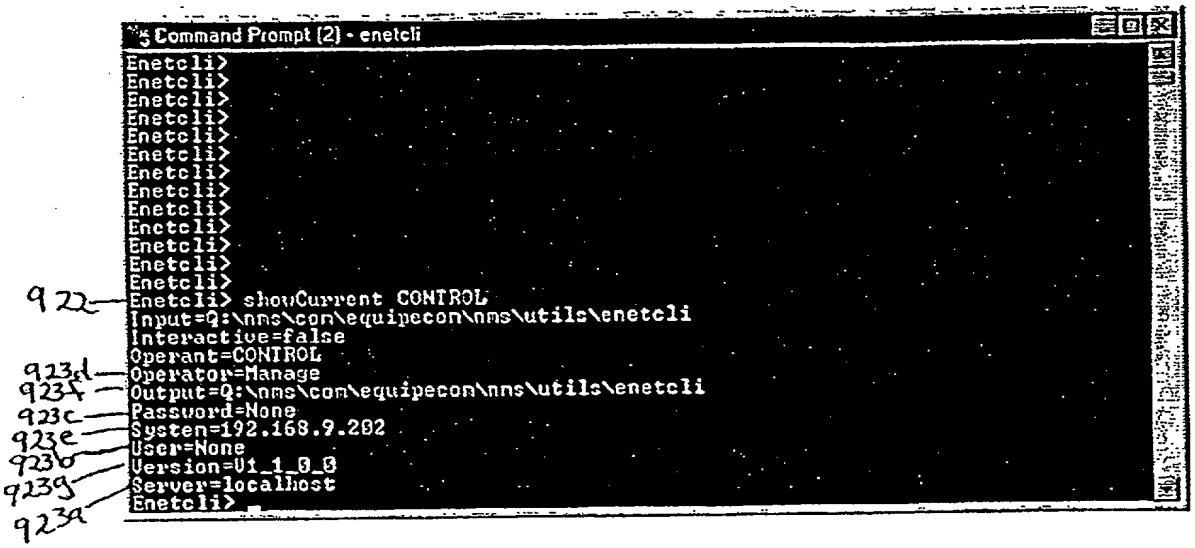

Open ?
No
921a
Yes
OSS client connects to NMS server
921b
manage ?
No
921c
Yes
OSS issues call to NMS server
921d
NMS server opens connection with network device
921e
execute ?
No
921f
Yes
OSS executes template instructions and sends calls to NMS server
921g
NMS server receives calls and completes provisioning task
921h

Fig. 3K



09/15/2000 09:26:43.618

Fig. 3L

924 BATCH.

Operant=BATCH
Operator=Execute
Version=V1_1_0_0
924a—Task1=execute-SPATH
924b—Task2=execute-PVC
924c—Task3=execute-SPVC
924d—Task4=load-SPVC-spvc1
924e—Task5=execute-SPVC
924f—Task6=load-SPVC-spvc2
924e—Task7=execute-SPVC
.
.
.
924g—Task50=set-SPATH-PortID-3
924h—Task51=execute-SPATH
924i—Task52=set-SPATH-SlotID-2
924j—Task53=execute-SPATH

confirm # 09666446

Fig. 3M

925

Operant=BATCH
Operator=Execute
Version=V1_1_0_0
925a Task1=execute-CONTROL
925b Task2=execute-SPATH
925c Task3=set-SPATH-PortID-3
925d Task4=execute-SPATH
.
.
.
925e Task61=set-CONTROL-System-192.168.9.201
925f Task62=execute-CONTROL
925g Task63=execute-SPATH
.
.
.
925h Task108=close
925i Task109=set-CONTROL-Server-Server1
925j Task110=set-CONTROL-System-192.168.8.200
925k Task111=execute-CONTROL
925l Task112=execute-SPATH
.
.
.

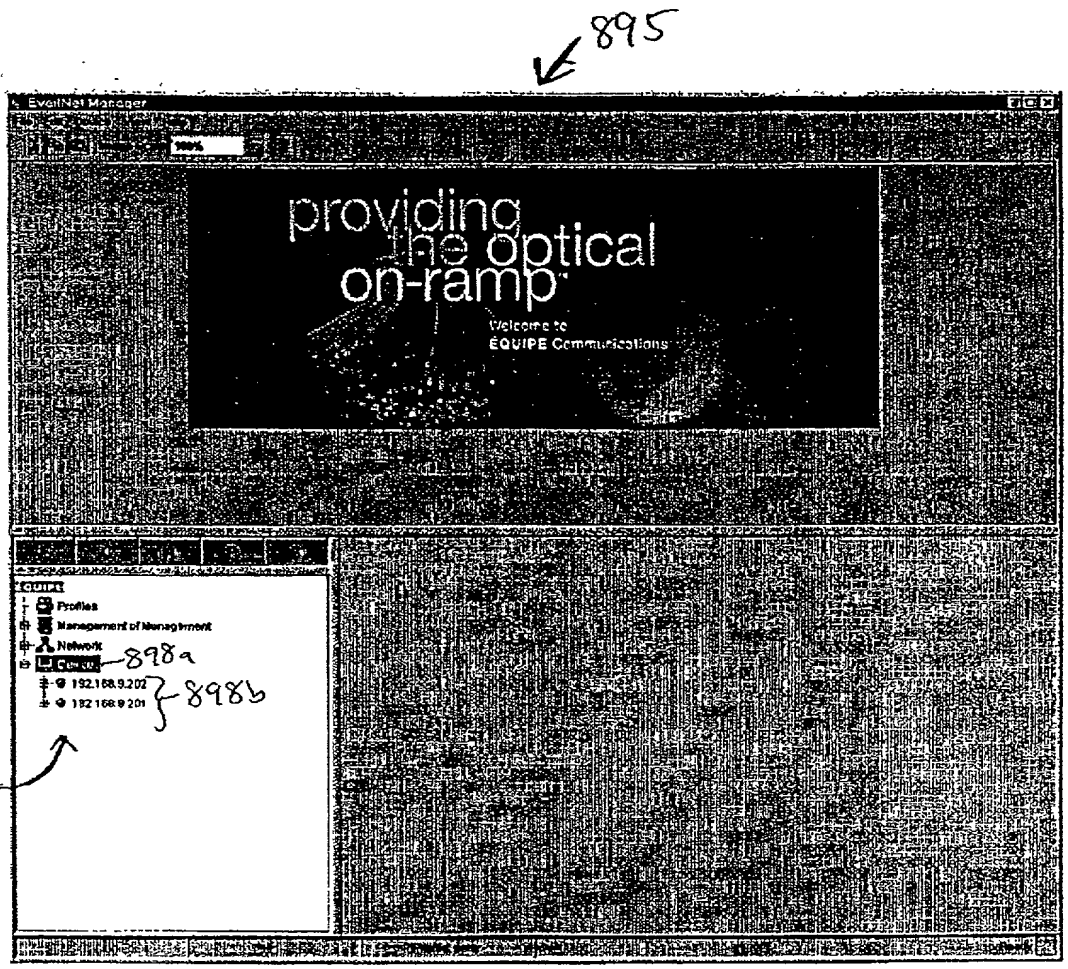

895
EquiNet Manager
providing the optical on-ramp
Welcome to
EQUIPE Communications
Profiles
Management of Management
Network
Console
192.168.9.202
192.168.9.201
898a
898b
898

Fig. 4a

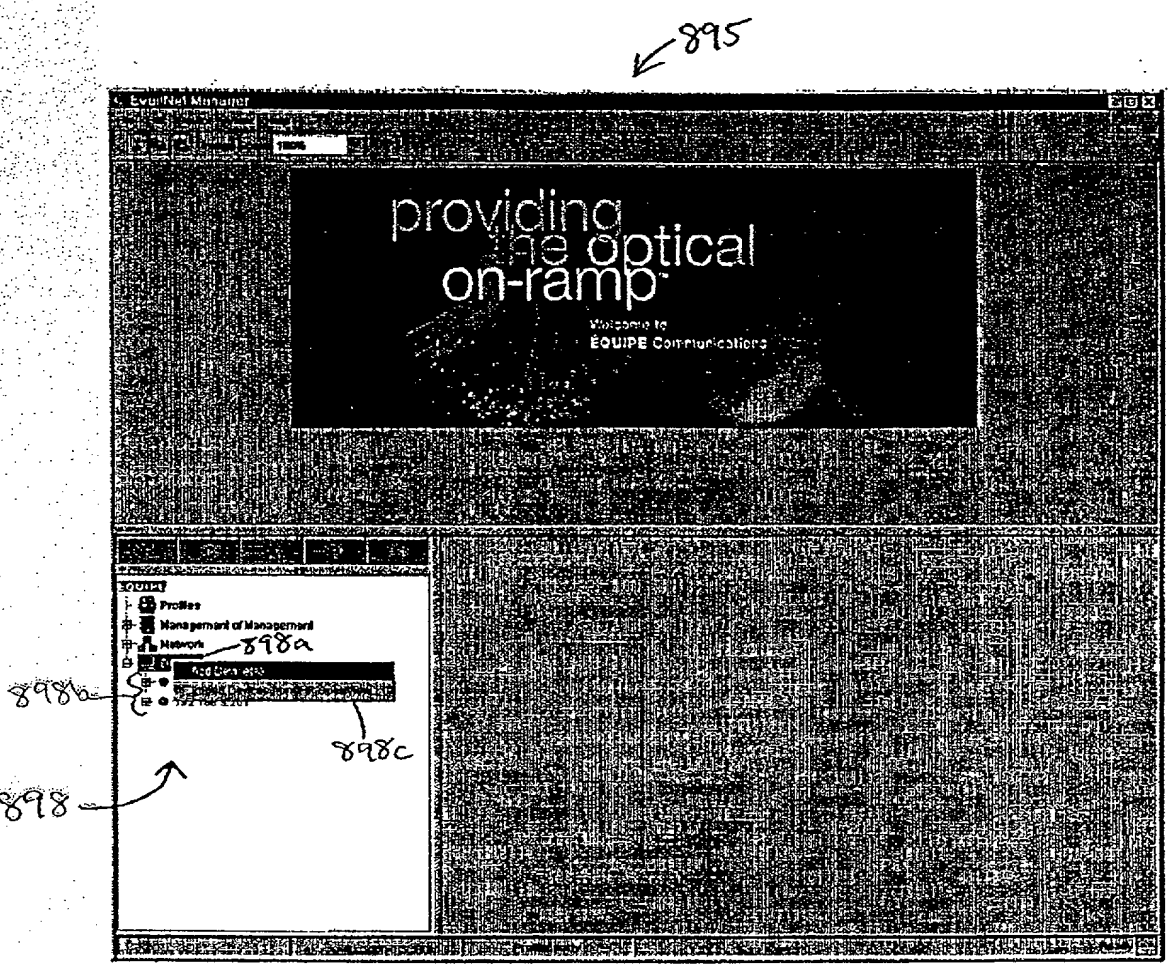

895
providing
the optical
on-ramp
EQUIPE Communications
898a
898b
898c
898

Fig. 4b

Fig. 4c

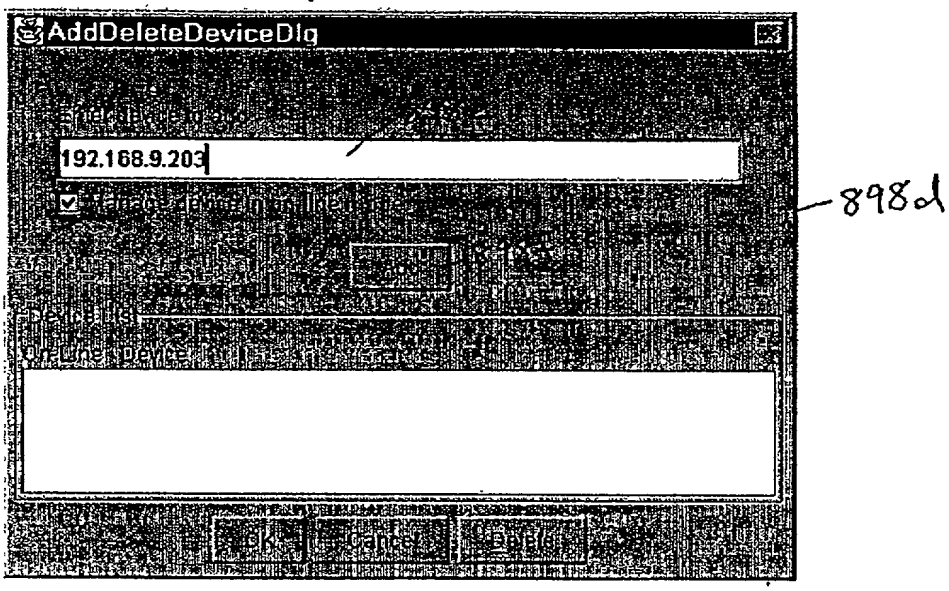
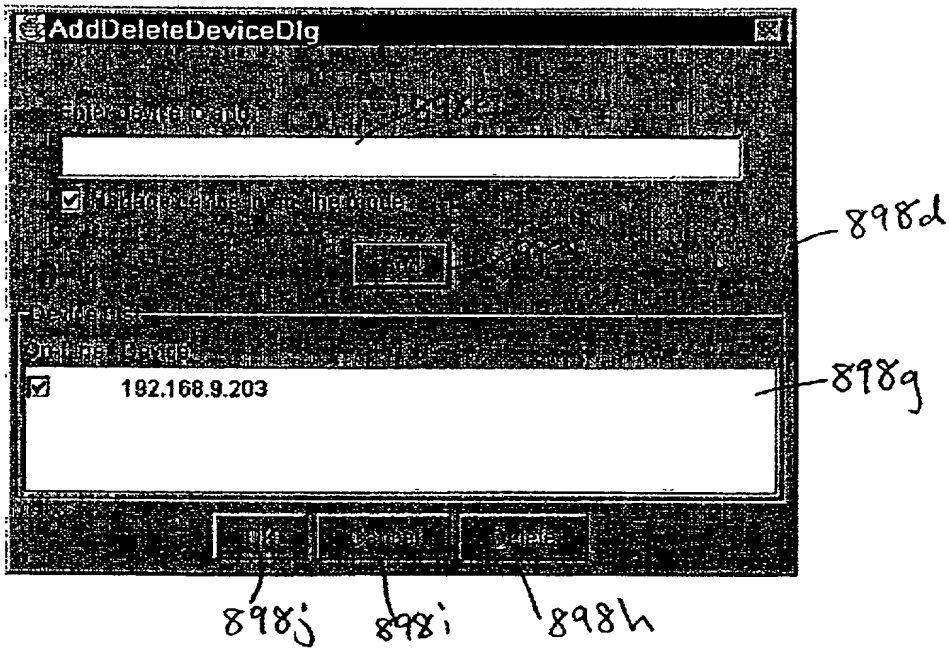

AddDeleteDeviceDlg
192.168.9.203
898d

Fig. 4d


AddDeleteDeviceDlg
898d
192.168.9.203
898g
898j
898i
898h

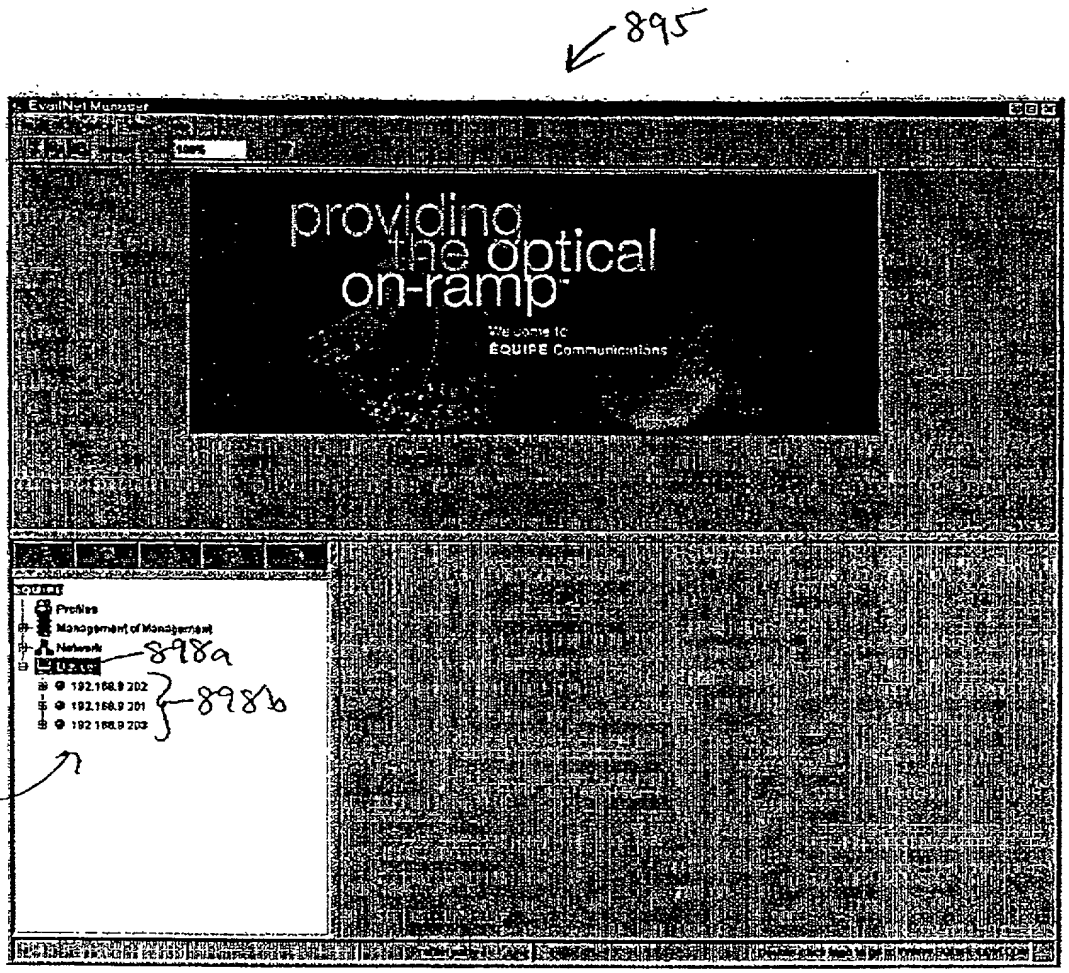

895
providing the optical on-ramp
Welcome to
EQUIPE Communications
Profiles
Management of Management
Network
898a
192.168.9.202
192.168.9.201
192.168.9.203
898b
898

Fig. 4e

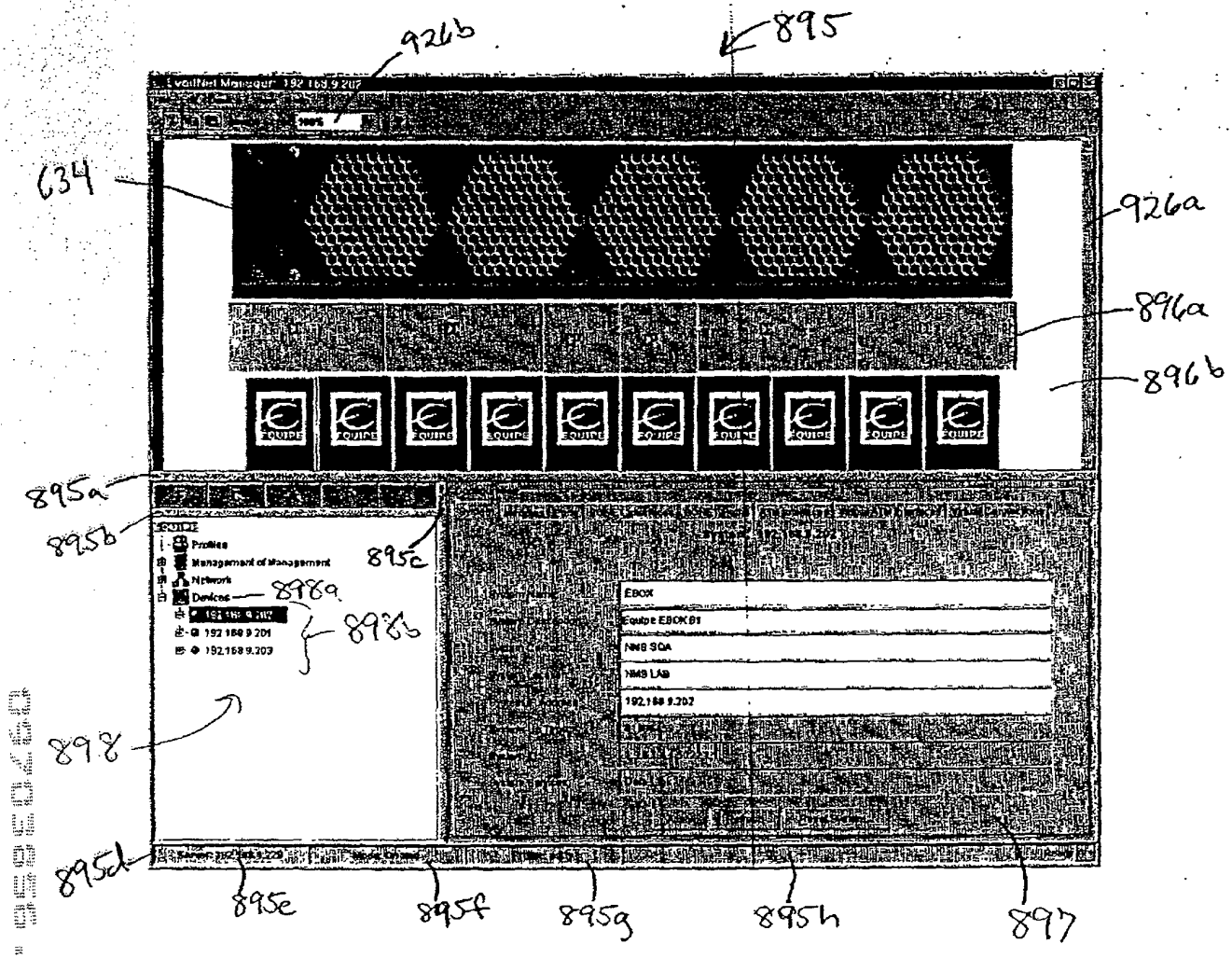

926b
895
634
926a
896a
896b
895a
895b
895c
898a
898b
898
895d
895e
895f
895g
895h
897
EBOX
Equipe EBOX 81
NMS SQA
NMS LAB

Fig. 4f

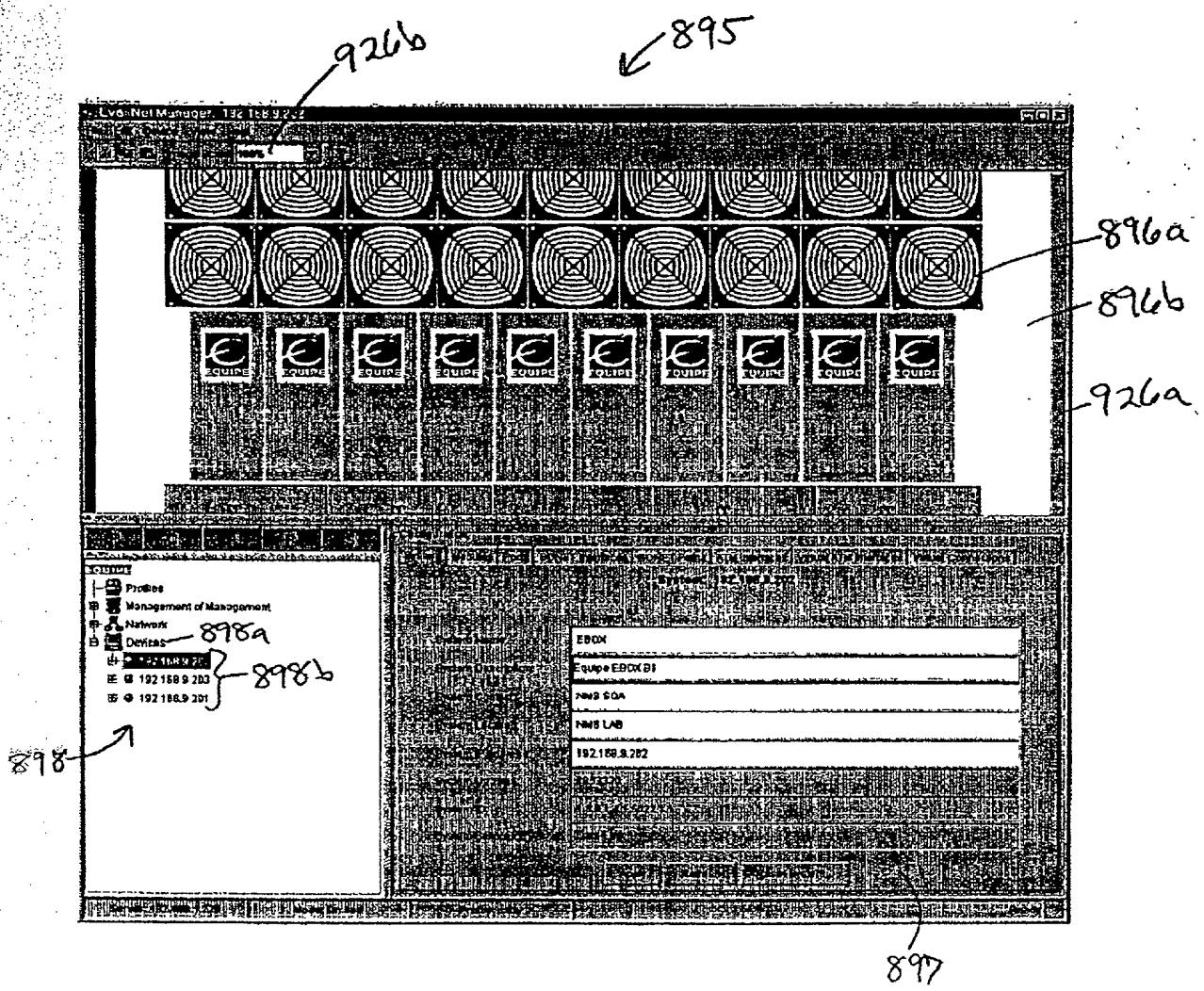

926b
895
896a
896b
926a
Probes
Management of Management
Network
Devices
898a
898b
898
EBDX
Equipe EBDX BI
NMS SQA
NMS LAB
192.168.9.202
897

Fig. 4g

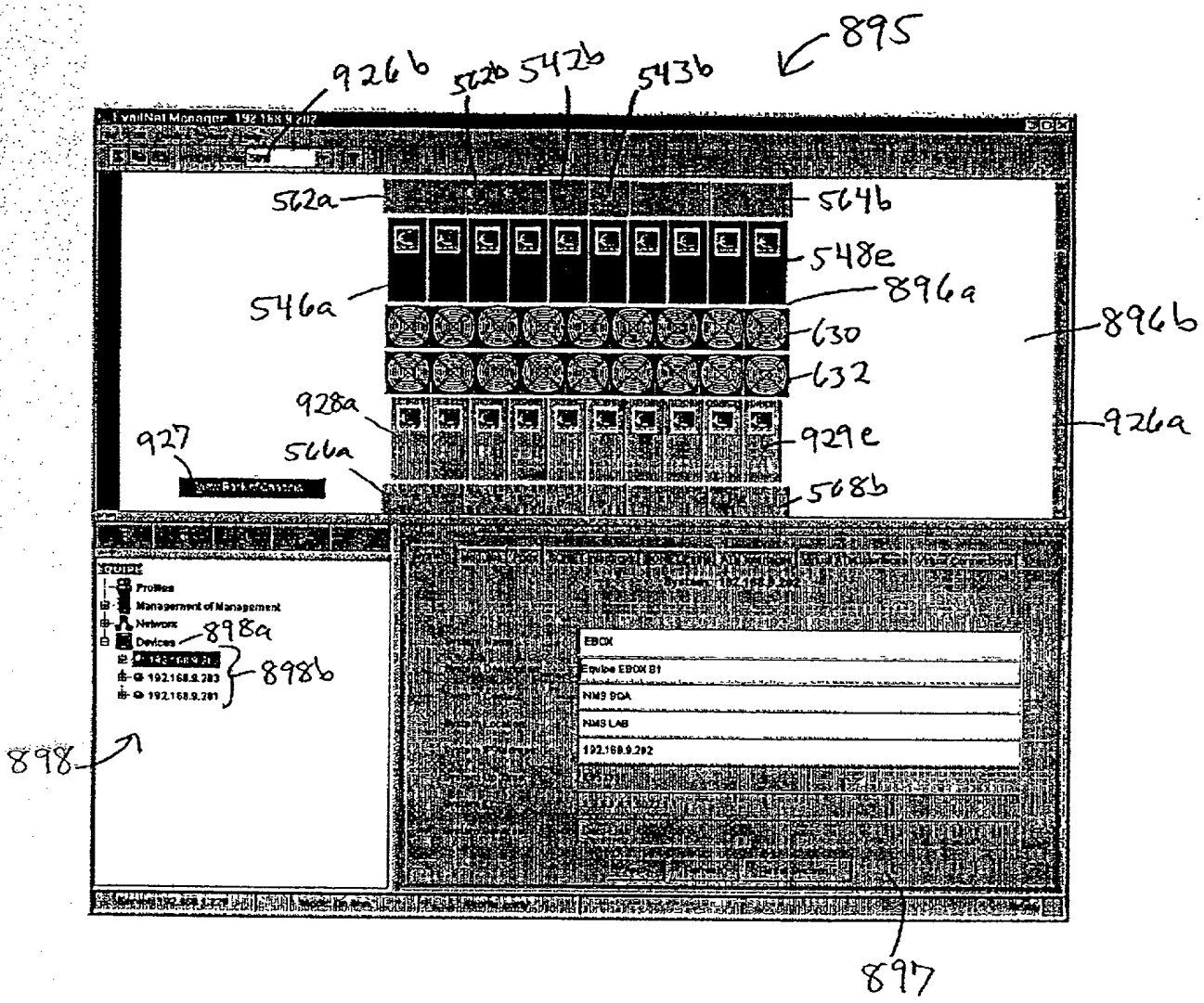

895
926b
520
542b
543b
562a
564b
548e
546a
896a
896b
630
632
928a
927
566a
929e
926a
568b
Profiles
Management of Management
Network
Devices
898a
898b
192.168.9.283
192.168.9.281
898
EBOX
Equipe EBOX B1
NMS SOA
NMS LAB
192.168.9.282
897

Fig. 4h

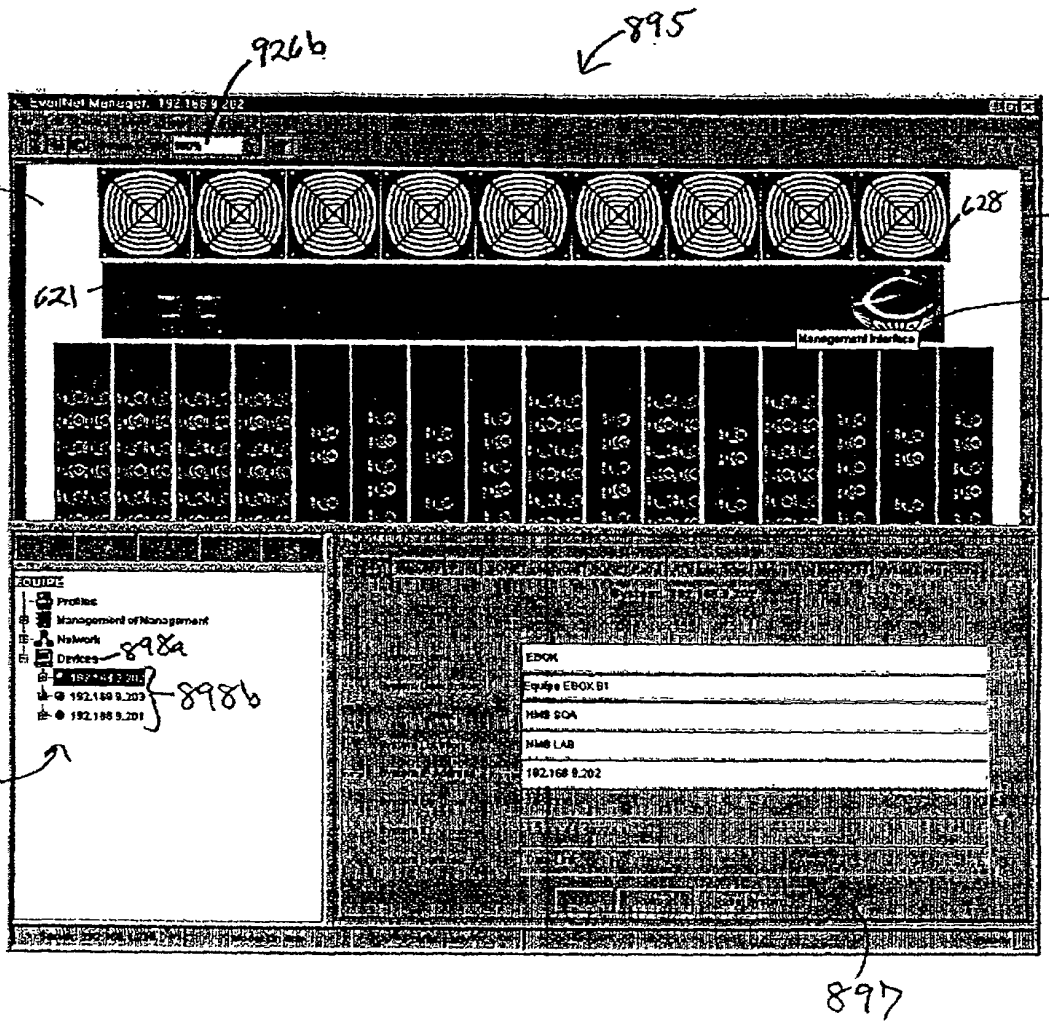


Fig. 4i

DEFENDANT'S EXHIBIT 1008

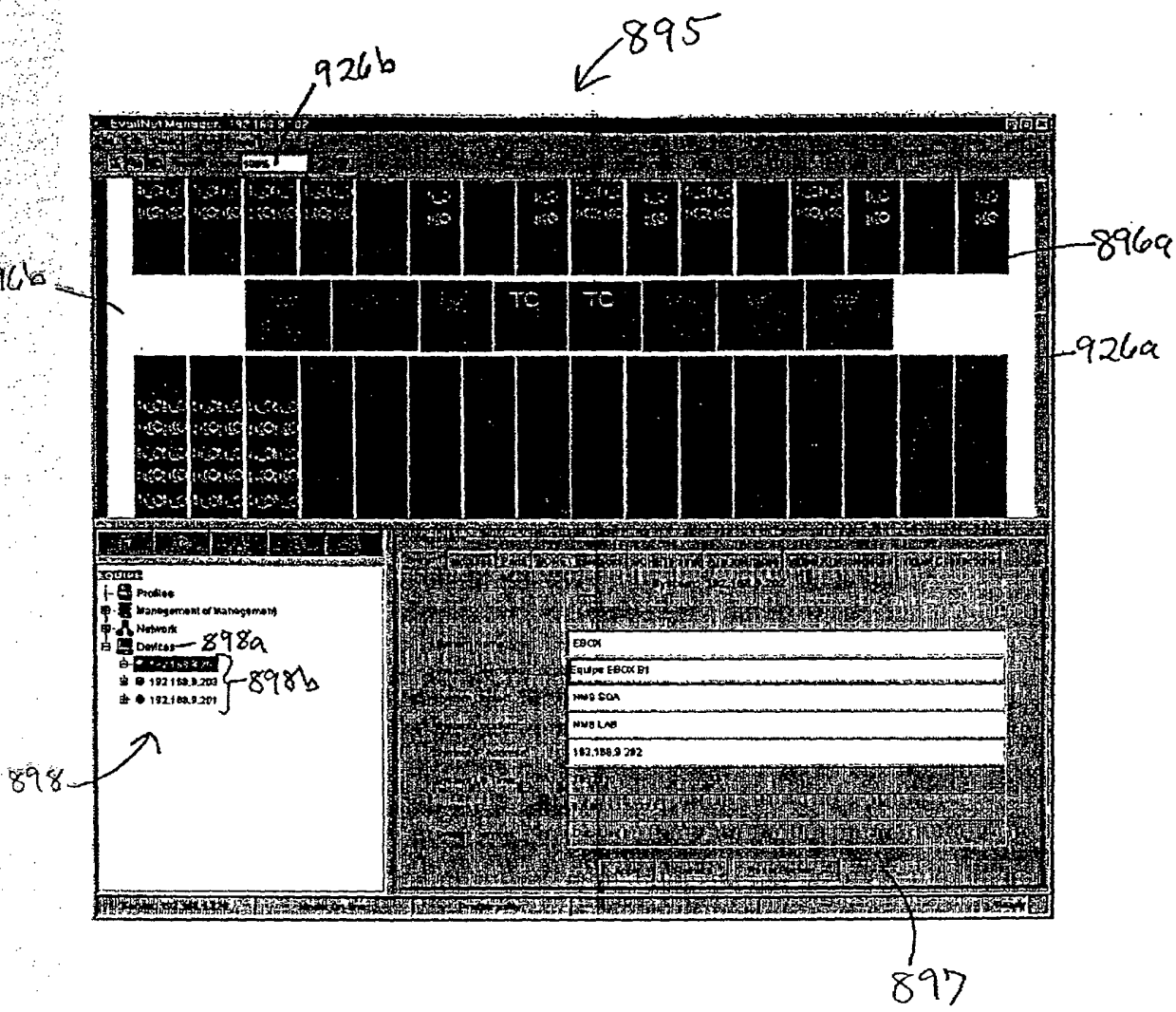

895
926b
896a
896a
926a
898
898a
898b
897
Profiles
Management of Management
Network
Devices
EBOX
Equipe EBOX B1
NMS SQA
NMS LAB
192.168.9.202

Fig. 4j

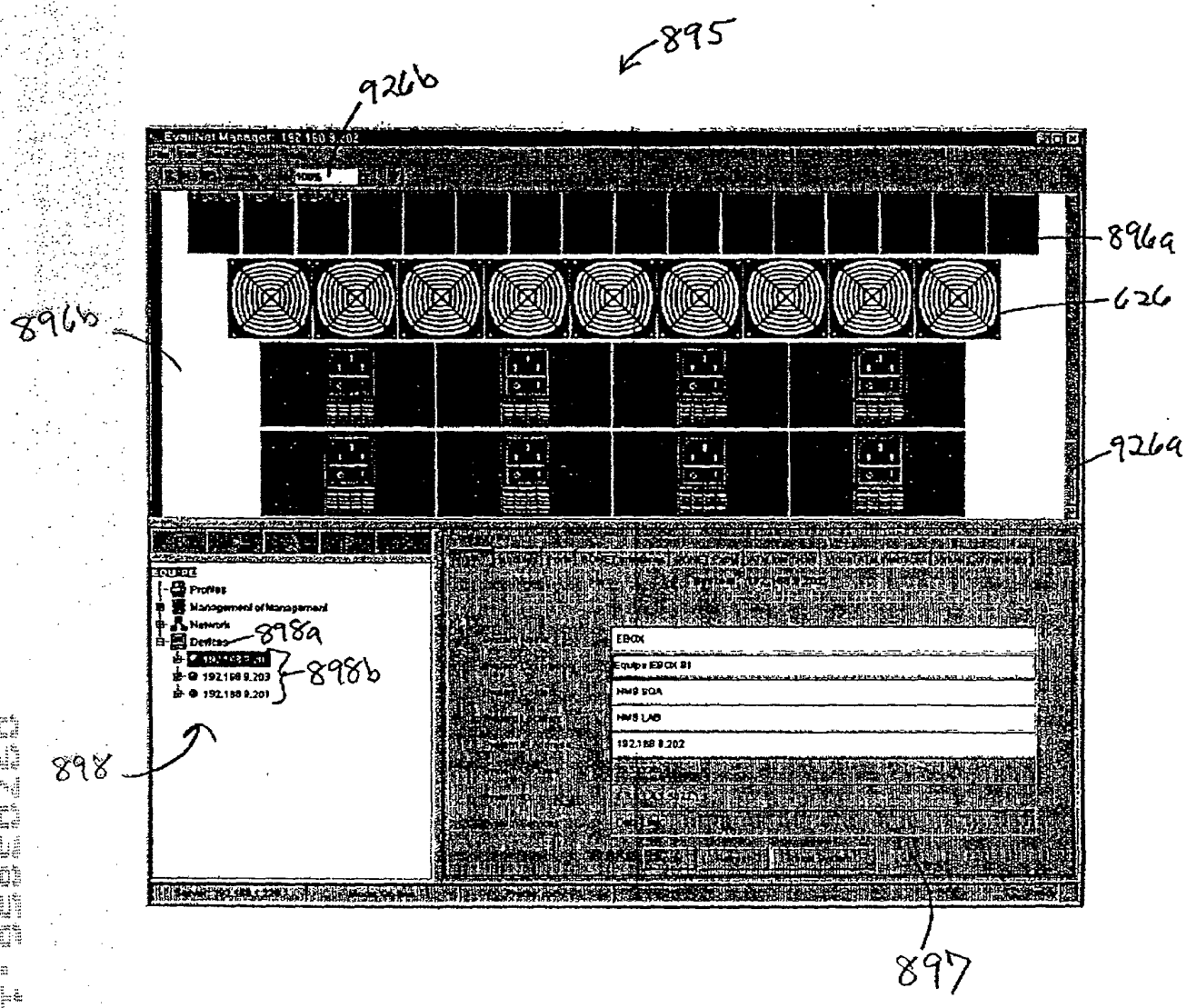

895
9266
896a
626
896b
9269
898a
898b
898
897
EBOX
Profiles
Management of Management
Network
Devices

Fig. 4K

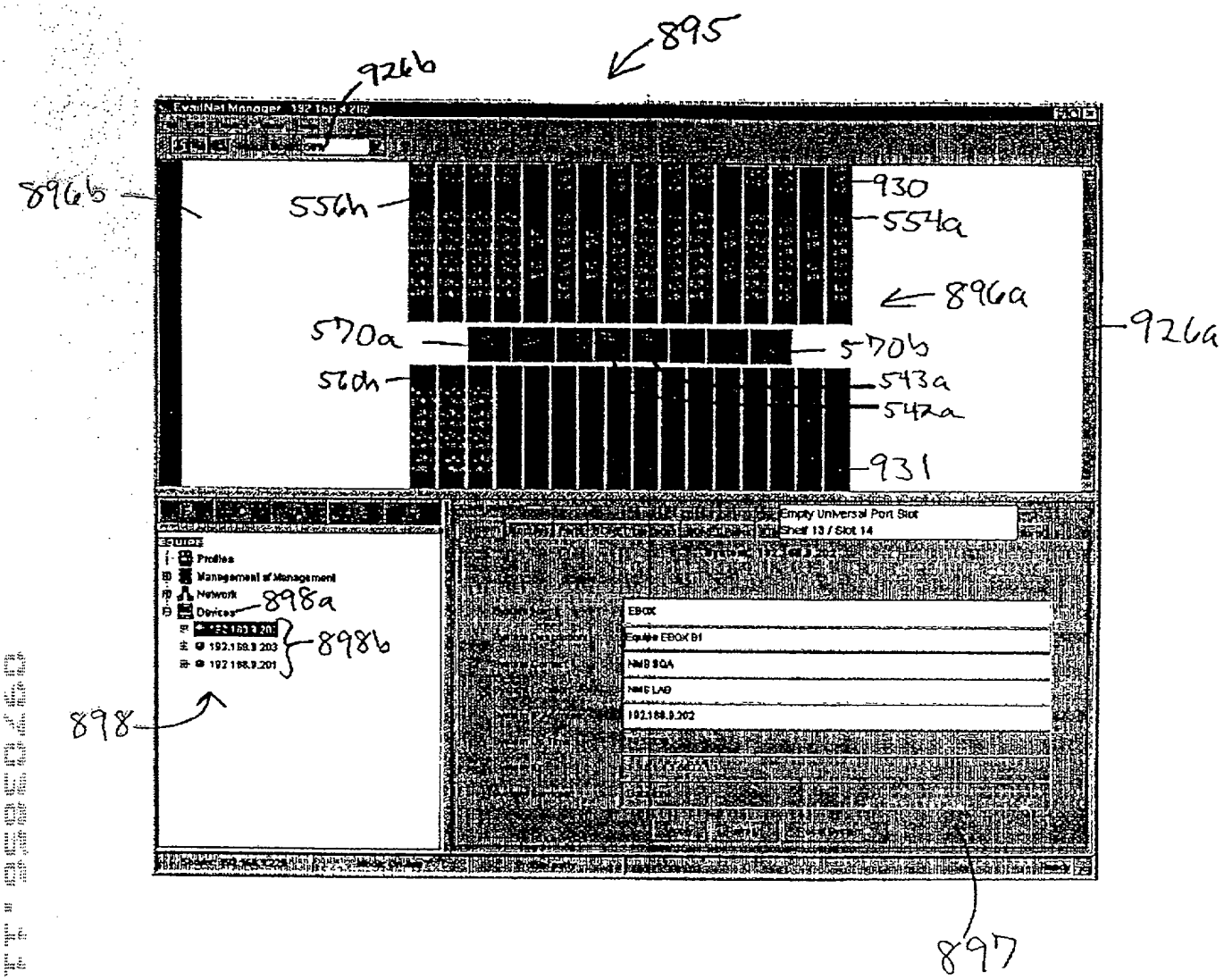

895
926b
896b
556h
930
554a
896a
926a
570a
570b
560h
543a
542a
931
Empty Universal Port Slot
Shelf 13 / Slot 14
898a
898b
898
897

Fig. 4L

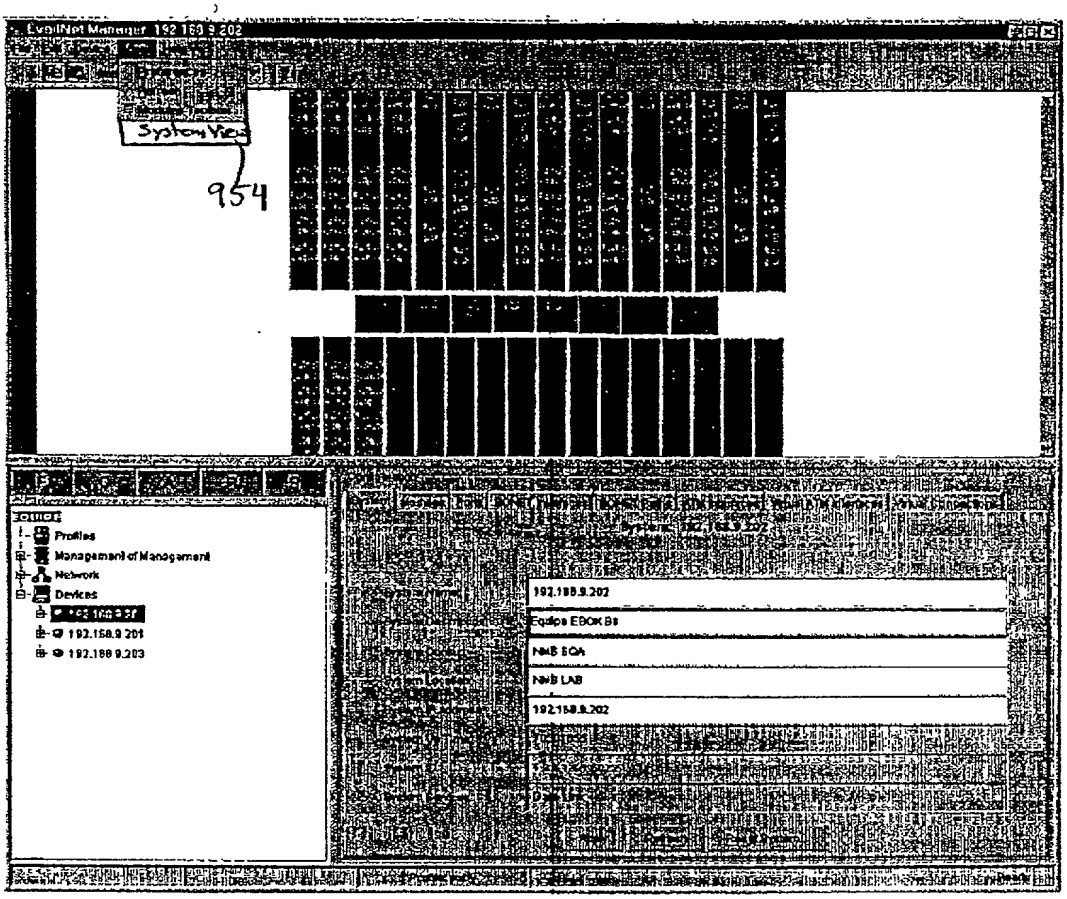

System View
954
Profiles
Management of Management
Network
Devices
192.168.9.201
192.168.9.203
192.168.9.202
Equipe EBOK B8
NetB EQA
NetB LAB
192.168.9.202

Fig. 4m

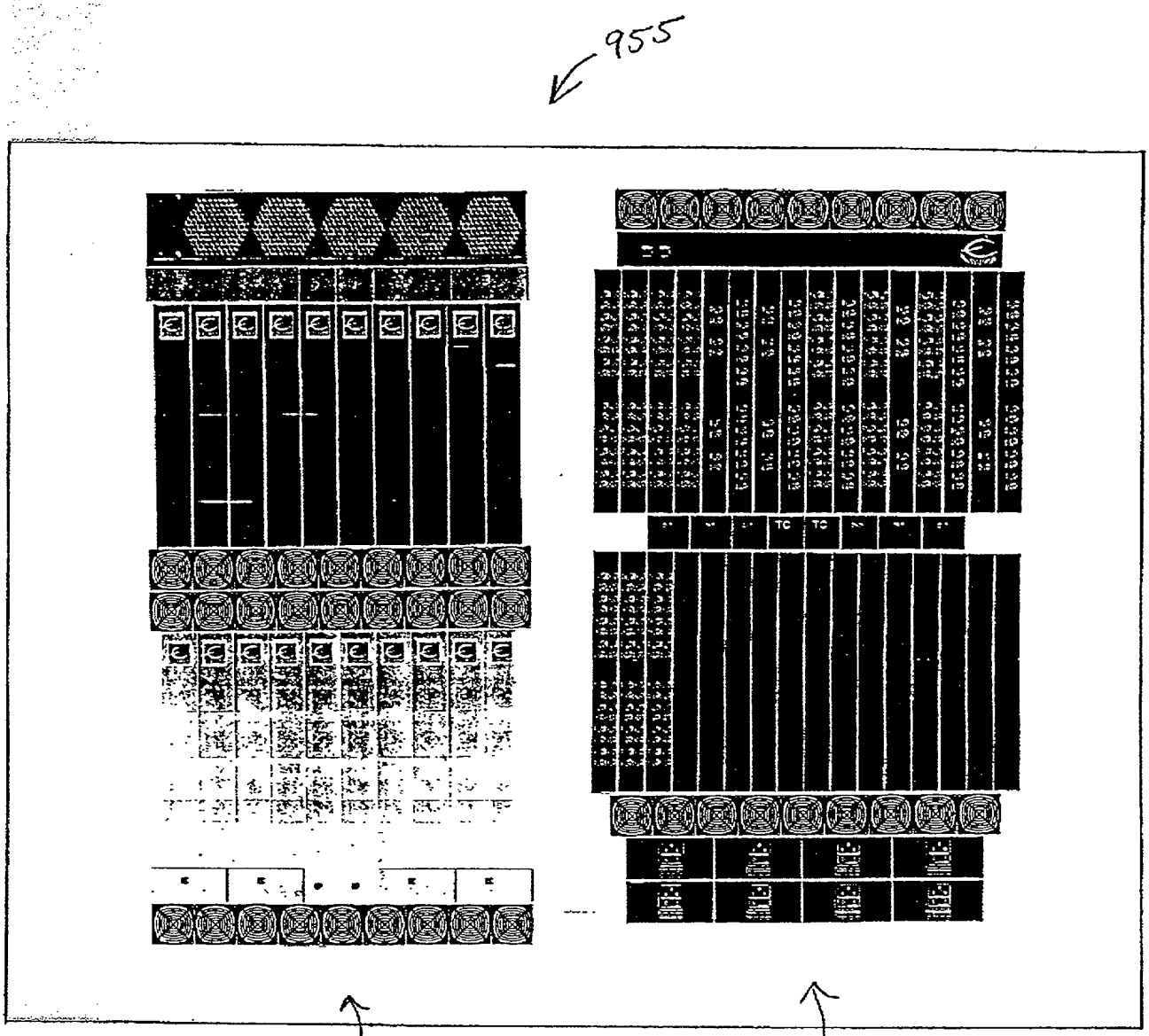


Fig. 4n

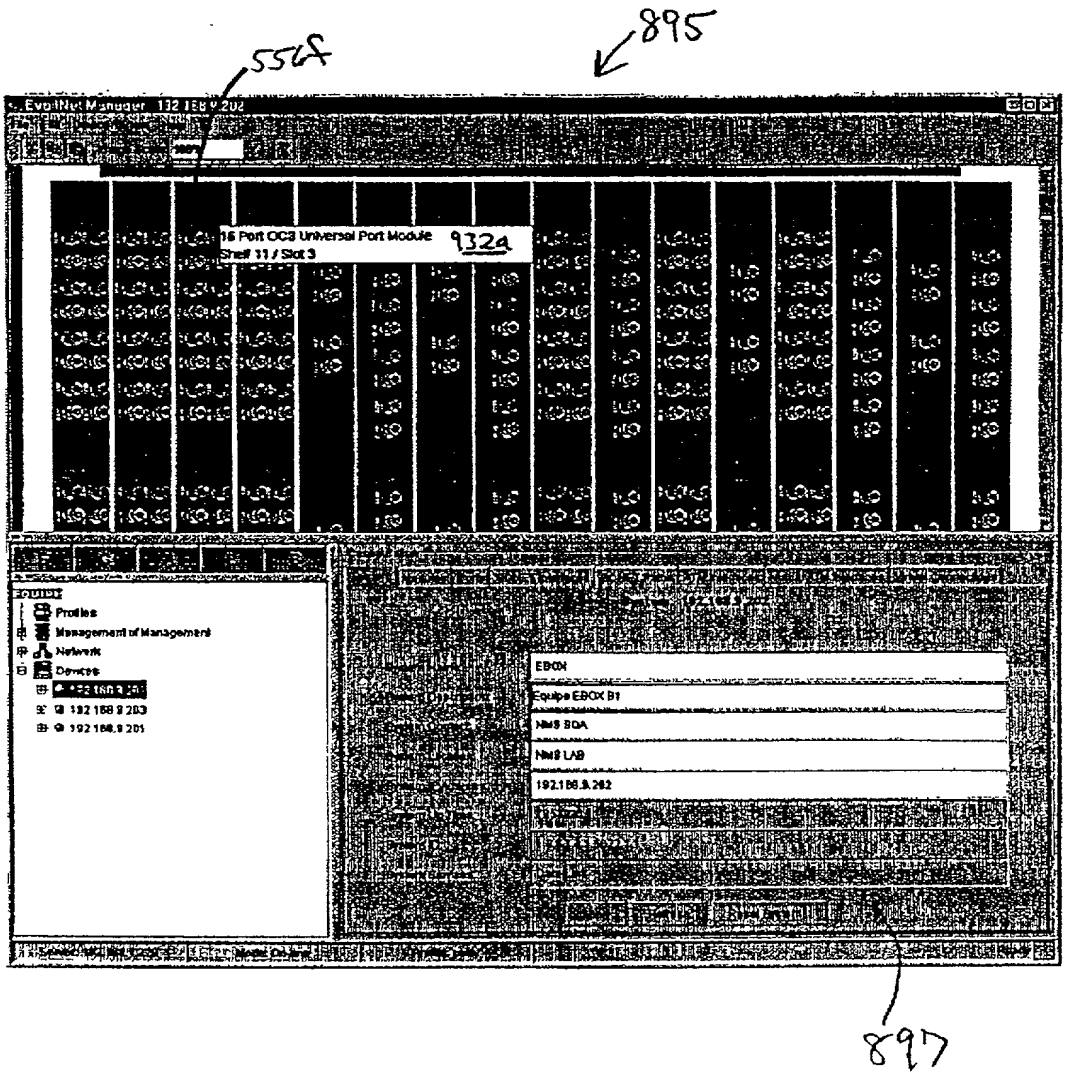

895
556
16 Port OC3 Universal Port Module
Shelf 11 / Slot 3
9324
Profiles
Management of Management
Network
Devices
EBOX
Equipe EBOX B1
NMS BOA
NMS LAB
192.168.9.202
897

Fig. 4o

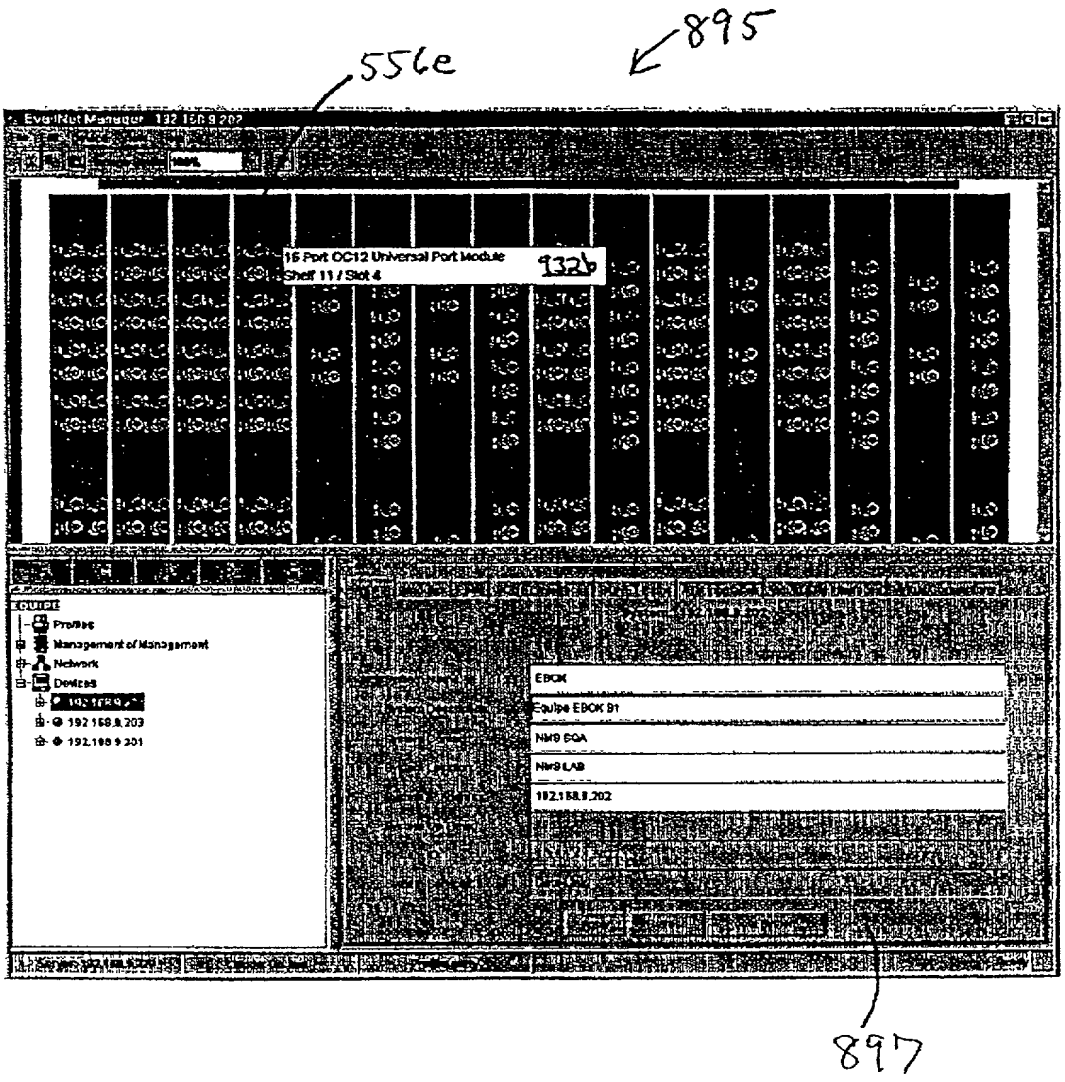

555e
895
16 Port OC12 Universal Port Module
Shelf 11 / Slot 4
932b
Profiles
Management of Management
Network
Devices
192.168.9.203
192.168.9.201
EBOK
Equipe EBOK 91
NMS EQA
NMS LAB
192.168.9.202
897

Fig. 4P

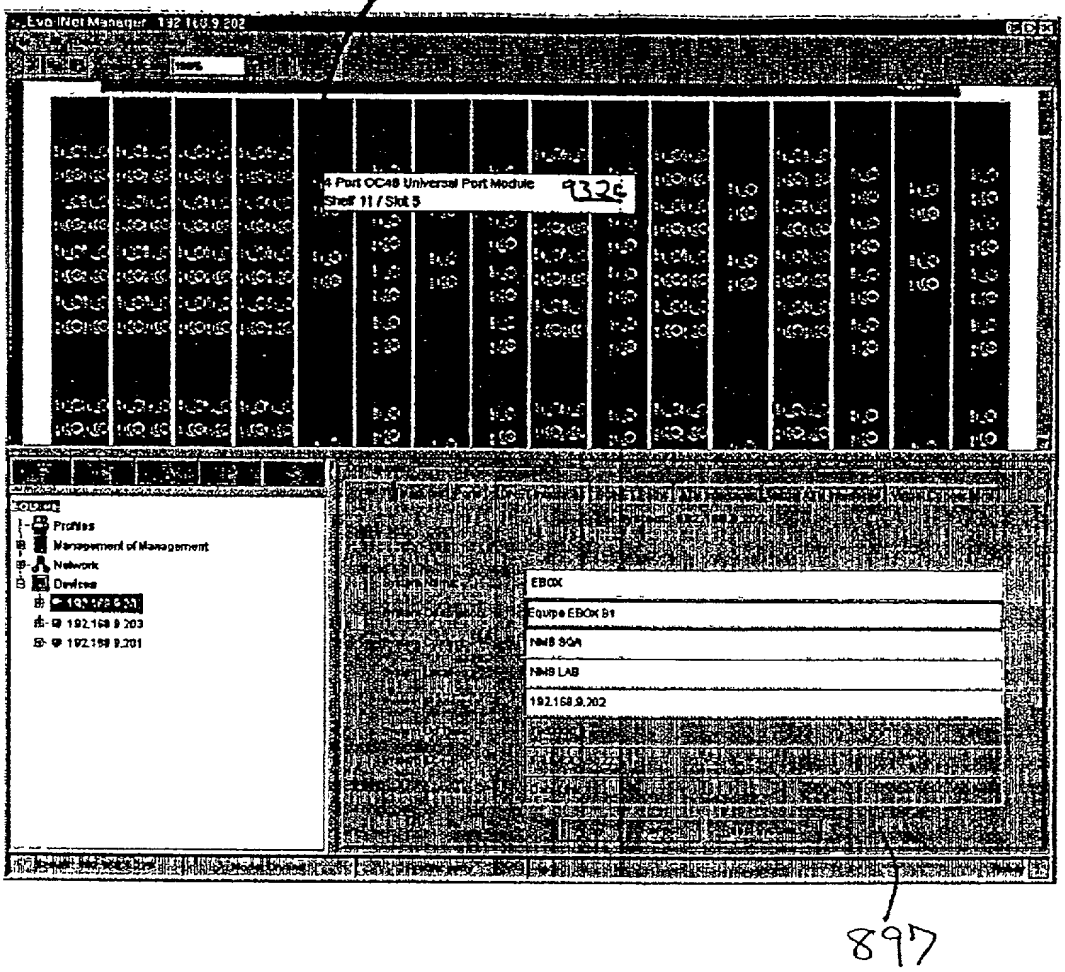

556d
895
4 Port OC48 Universal Port Module
Shelf 11 / Slot 5
932c
Profiles
Management of Management
Network
Devices
192.168.9.203
192.168.9.201
EBOX
Equipe EBOx B1
NMS SQA
NMS LAB
192.168.9.202
897

Fig. 4g

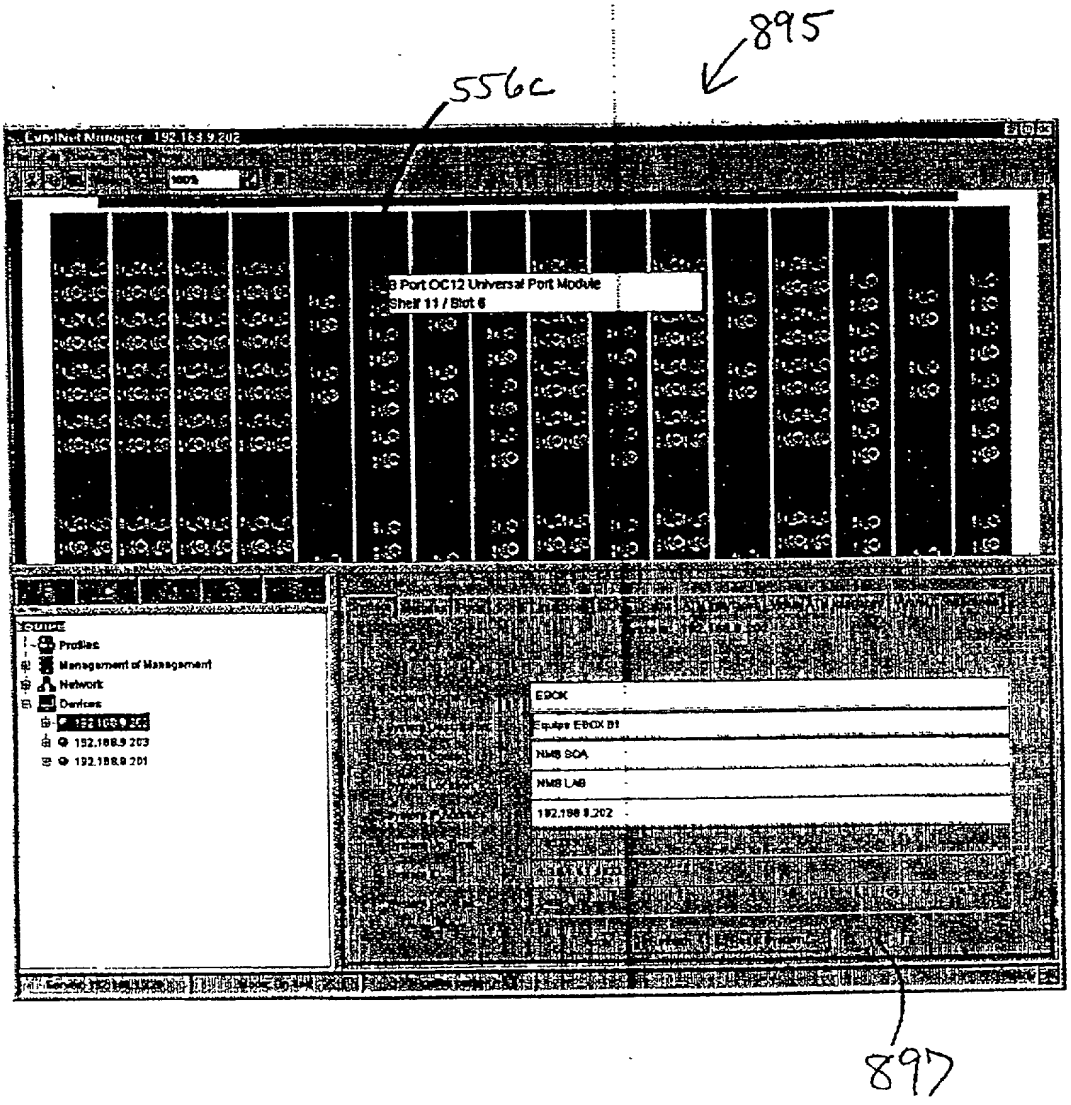

895
556c
8 Port OC12 Universal Port Module
Shelf 11 / Slot 6
Profiles
Management of Management
Network
Devices
192.168.9.203
192.168.9.201
EPOX
Equipe EBOX 01
NMS SOA
NMS LAB
192.168.9.202
897

Fig. 4F

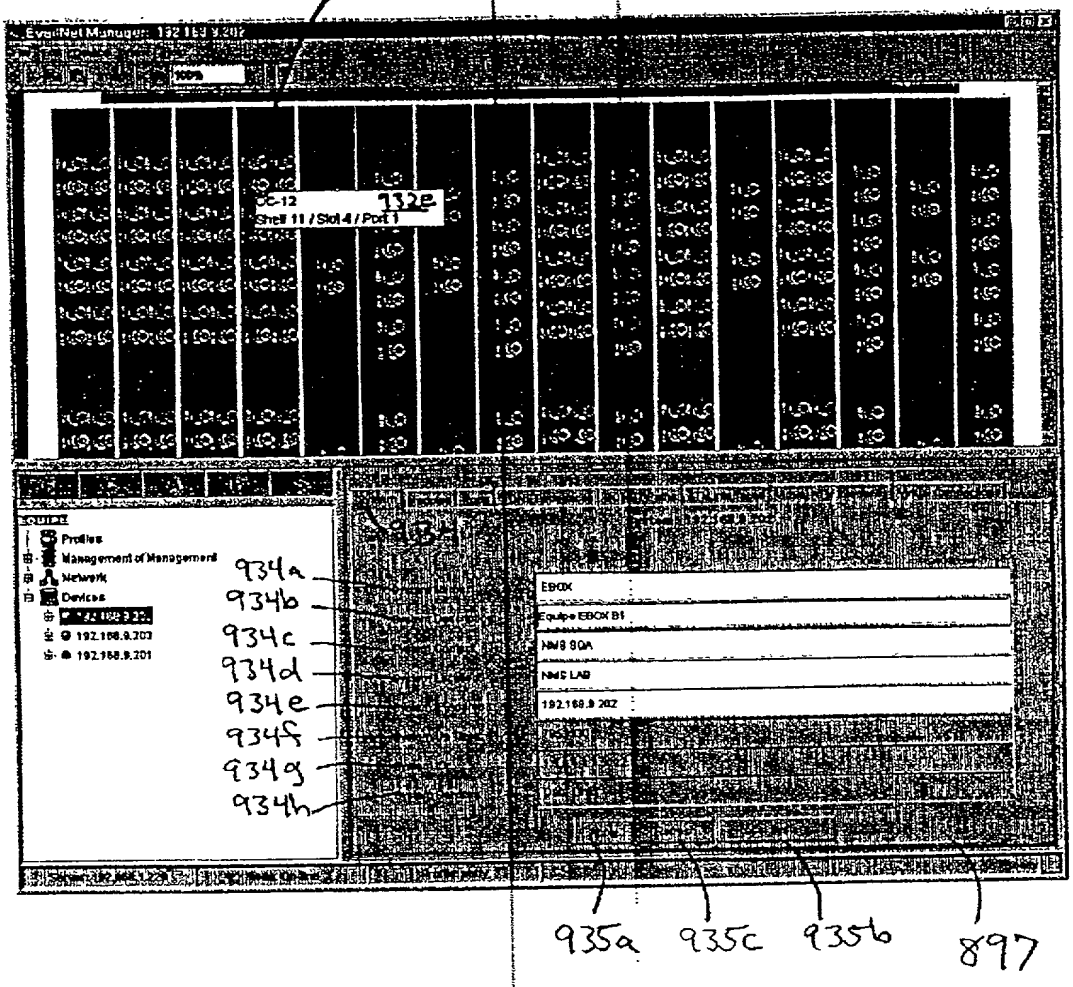

895
933
732e
934a
934b
934c
934d
934e
934f
934g
934h
935a
935c
935b
897

Fig. 4s

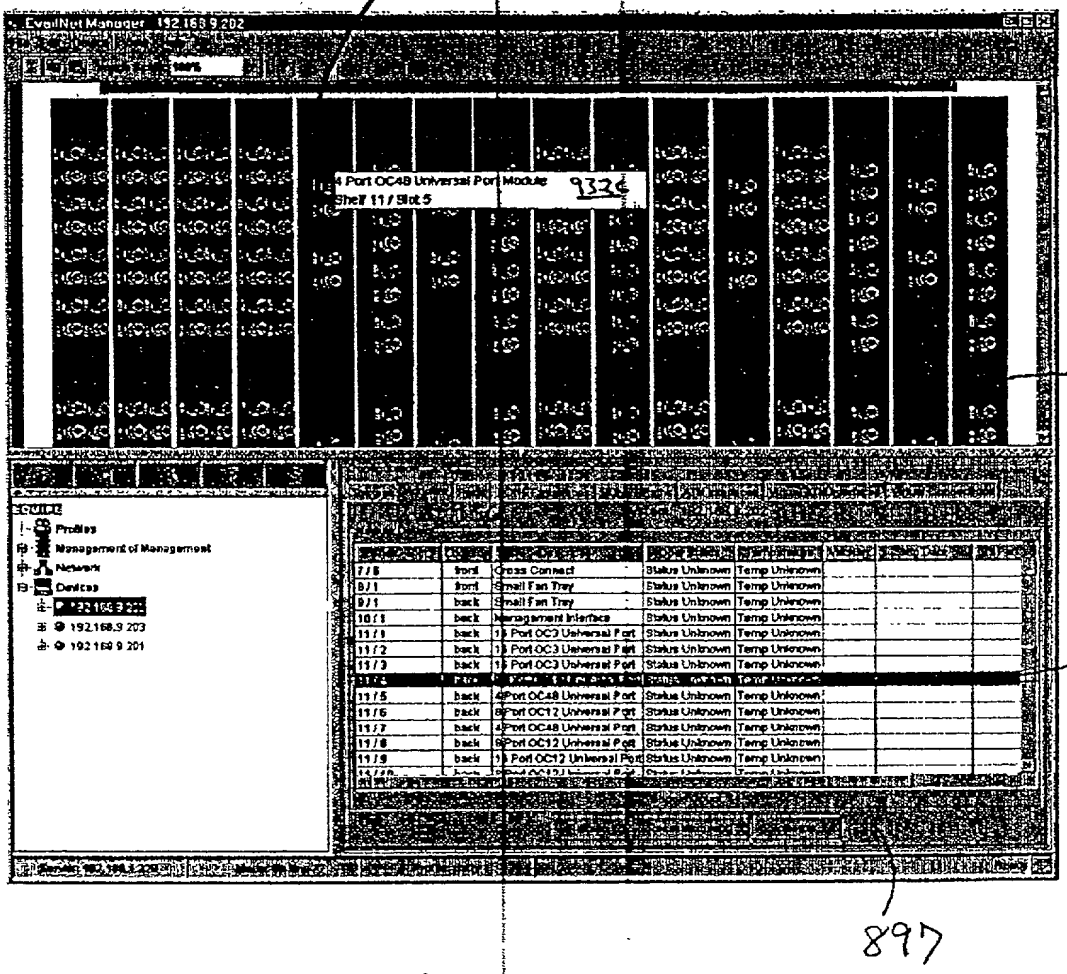


Fig. 44

PATENT APPLICATION

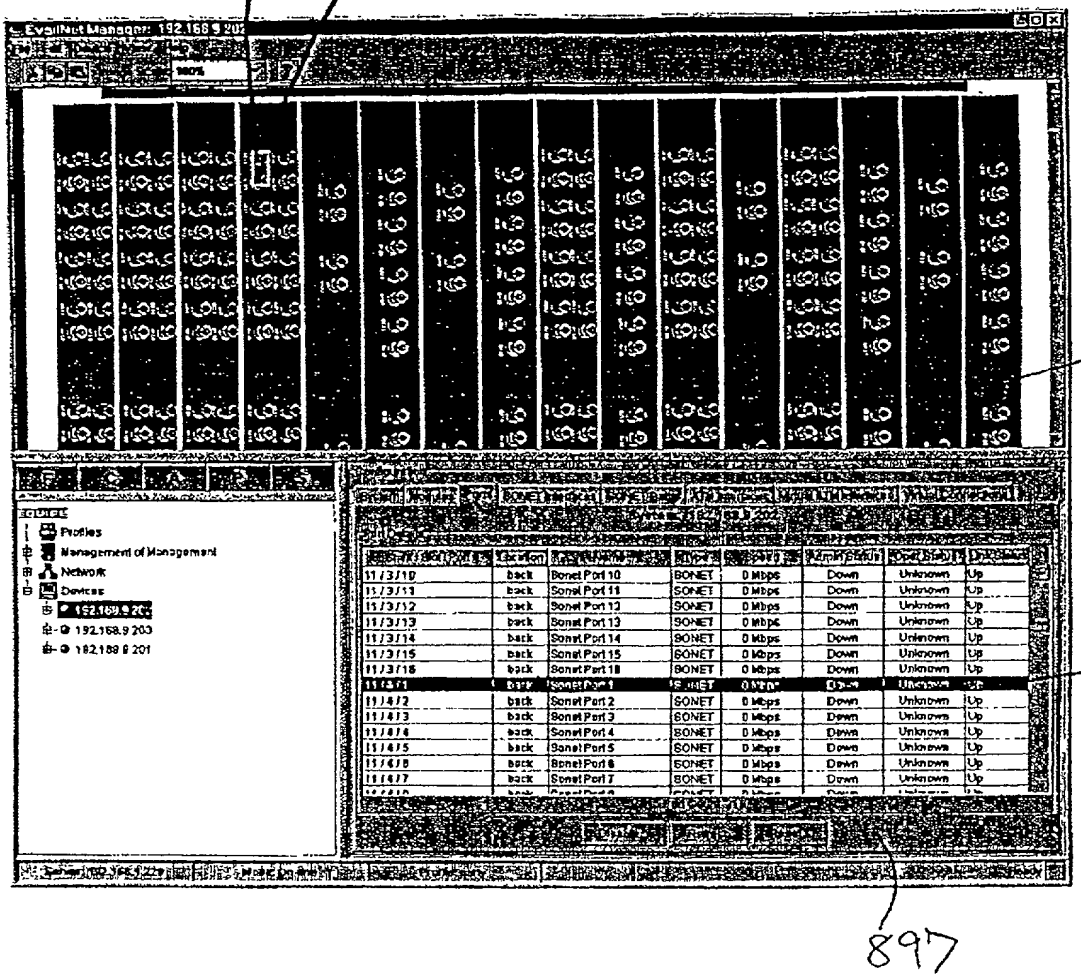

939a
556e
895
896a
939b
897
Profiles
Management of Management
Network
Devices
192.168.9.202
192.168.9.203
192.168.9.201

Fig. 4u

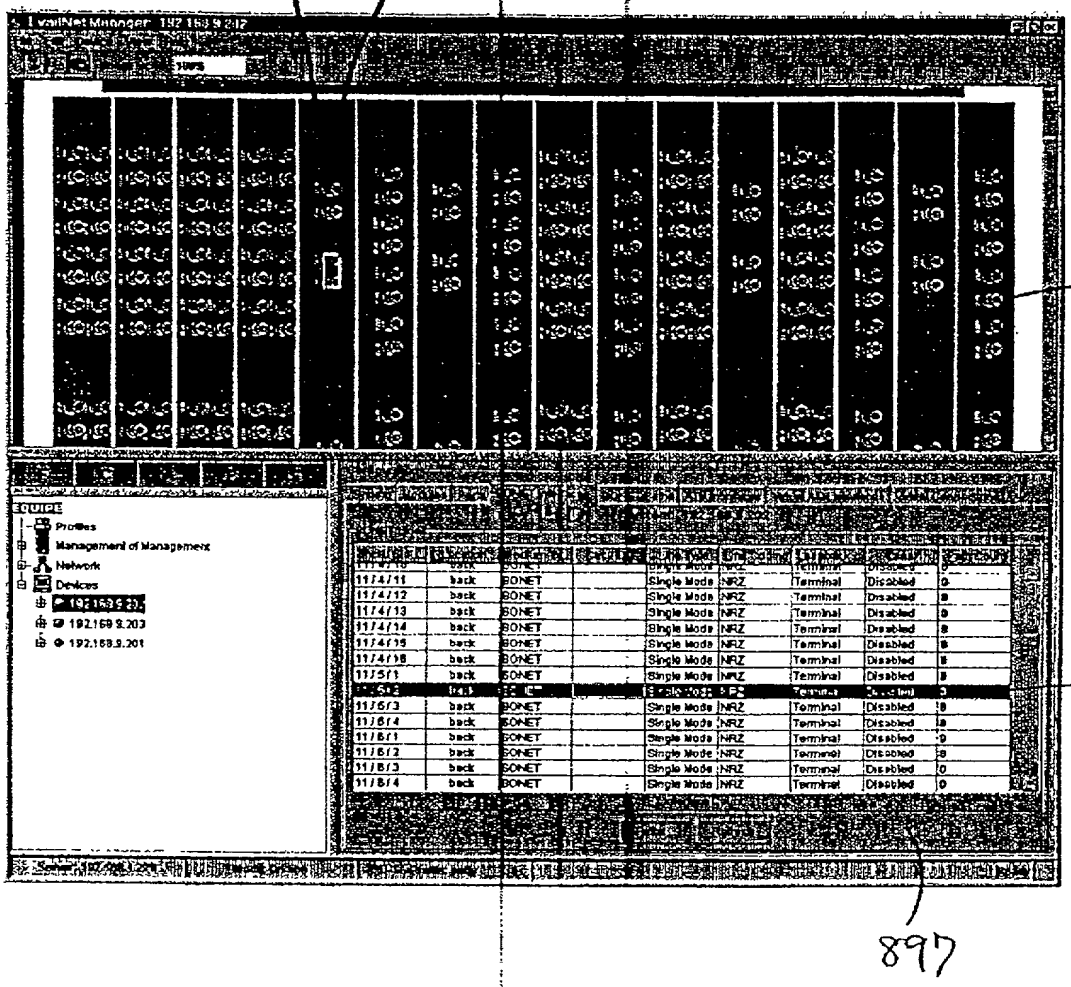

556d
941a
895
896a
941b
897
EQUIPE
Profiles
Management of Management
Network
Devices
192.168.9.203
192.168.9.201

Fig. 4v

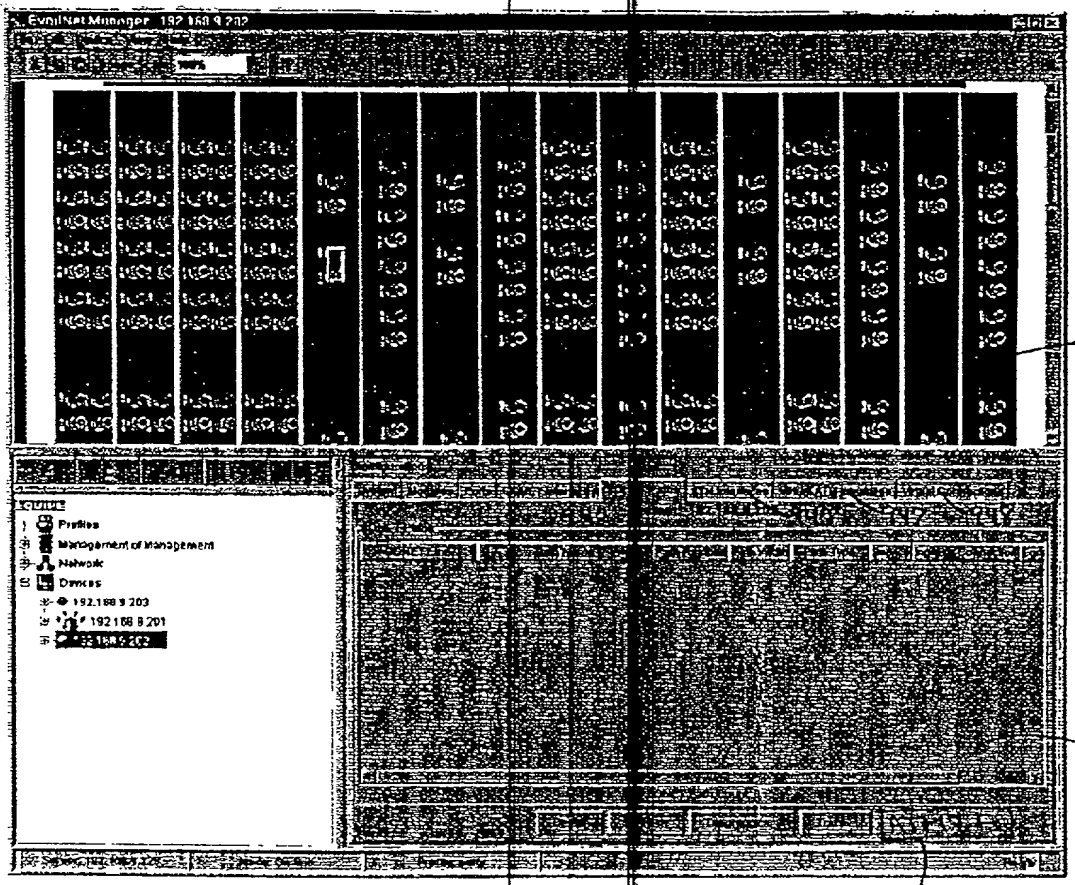


Fig. 4W

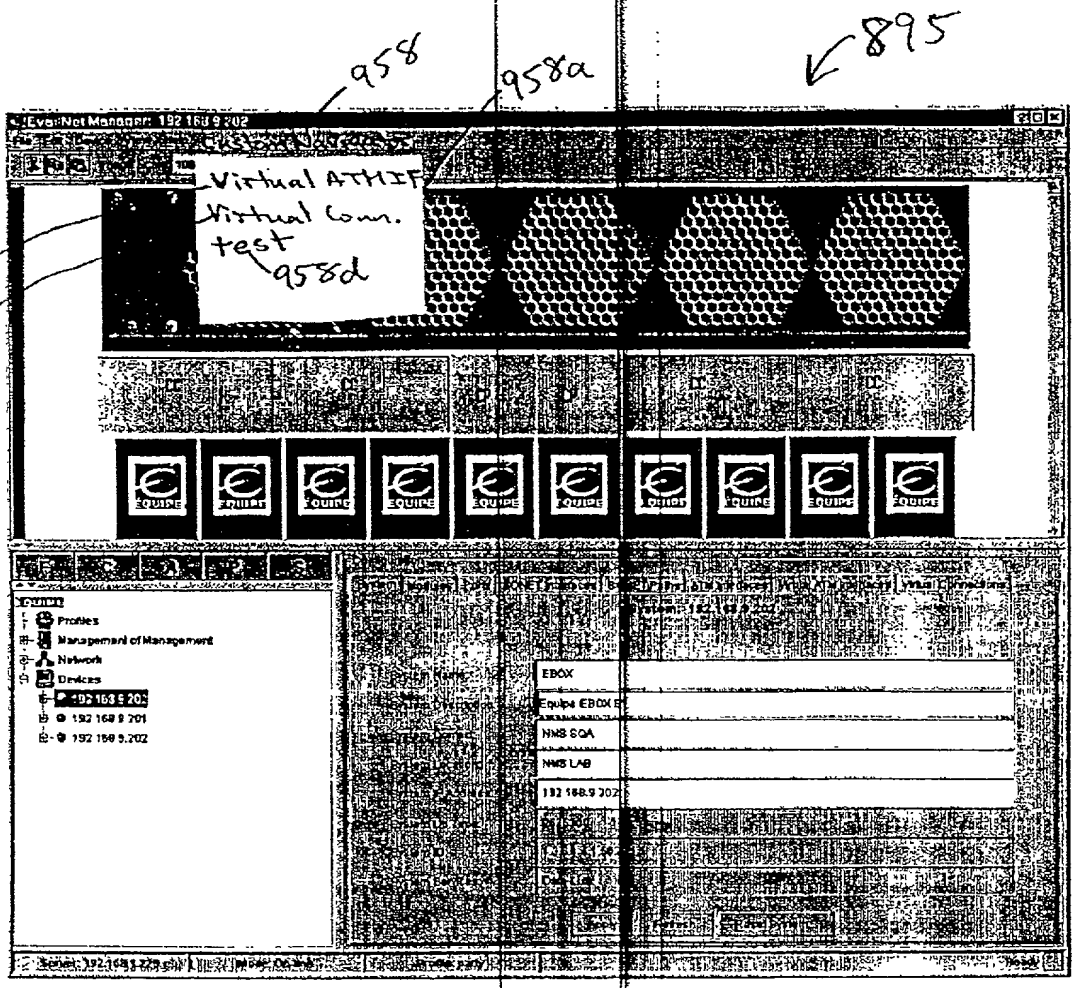

958
958a
895
Virtual ATHIP
Virtual Conn.
test
958d
958b
958c
Profiles
Management of Management
Network
Devices
EBOX
NMS SQA
NMS LAB

Fig. 4X

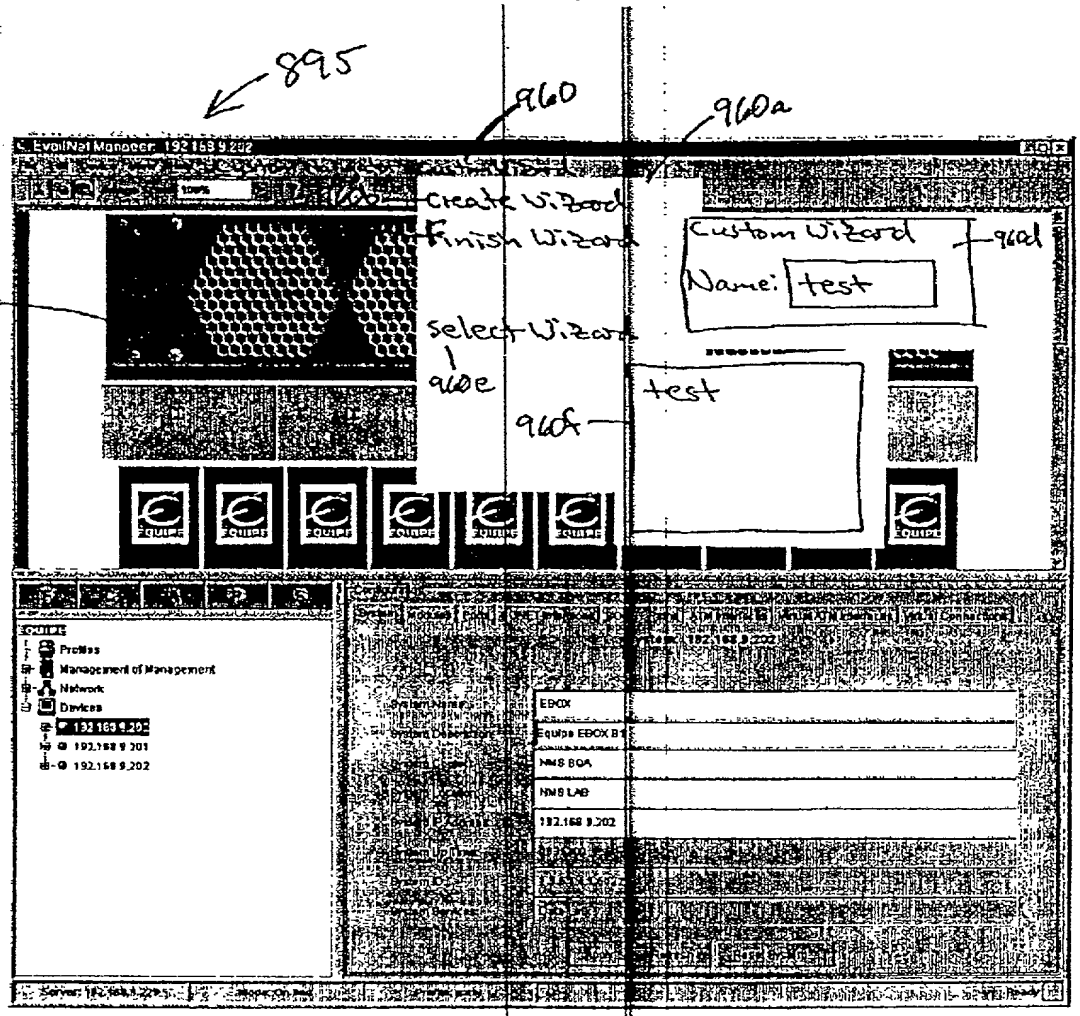

895
960
960a
EvalNet Manager 192.168.9.202
Create Wizard
Finish Wizard
Select Wizard
960e
Custom Wizard
960d
Name:
test
test
896a
Profiles
Management of Management
Network
Devices
192.168.9.202
192.168.9.201
192.168.9.202
EBOX
Equipe EBOX B1
NMS BQA
NMS LAB
192.168.9.202

Fig. 4y

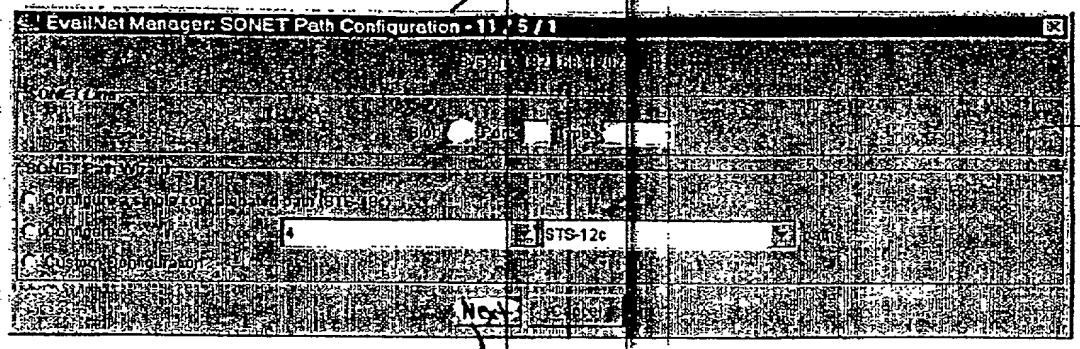

961
EvailNet Manager: SONET Path Configuration - 11 . 5 / 1
961a
STS-12c
961b

Fig. 42

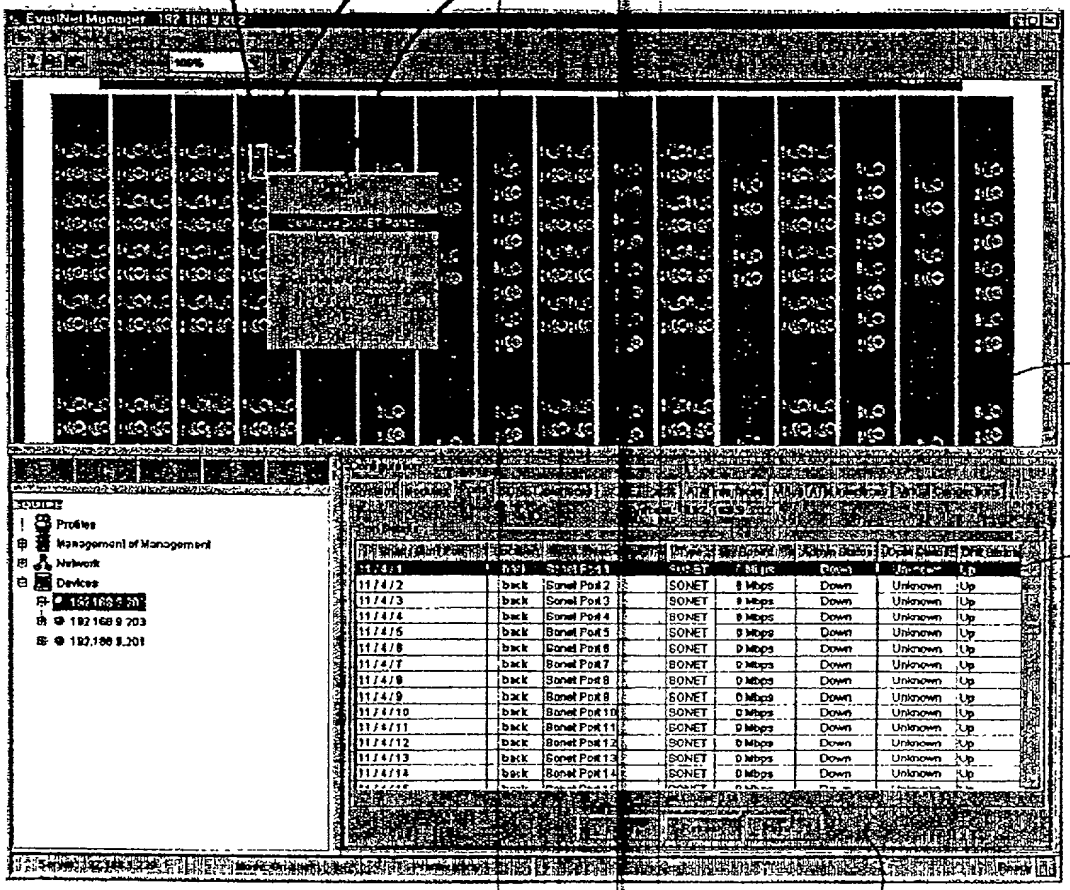


Fig. 5a

DRAFT - 08/29/00 - 12:45 PM

Fig. 5b

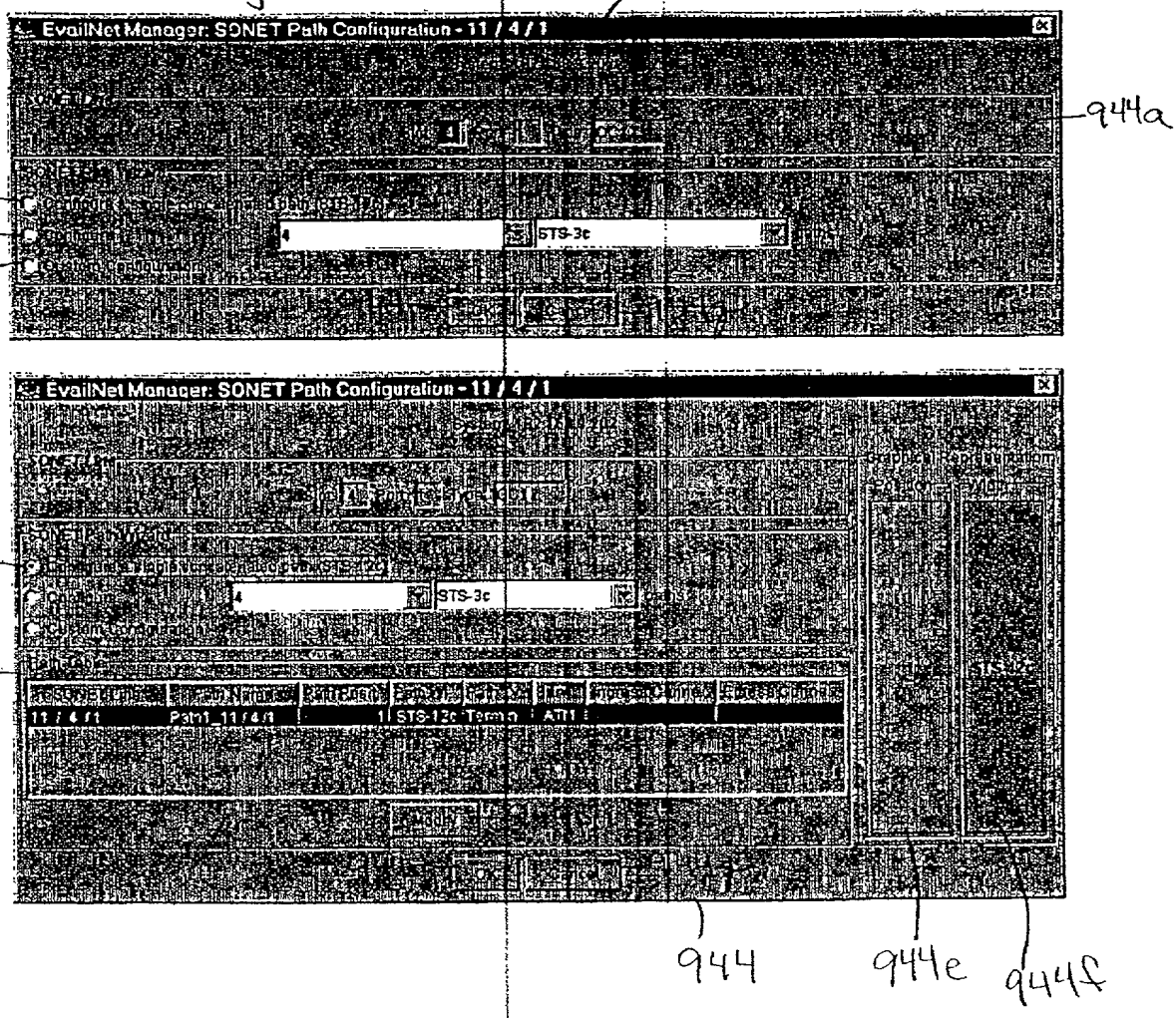
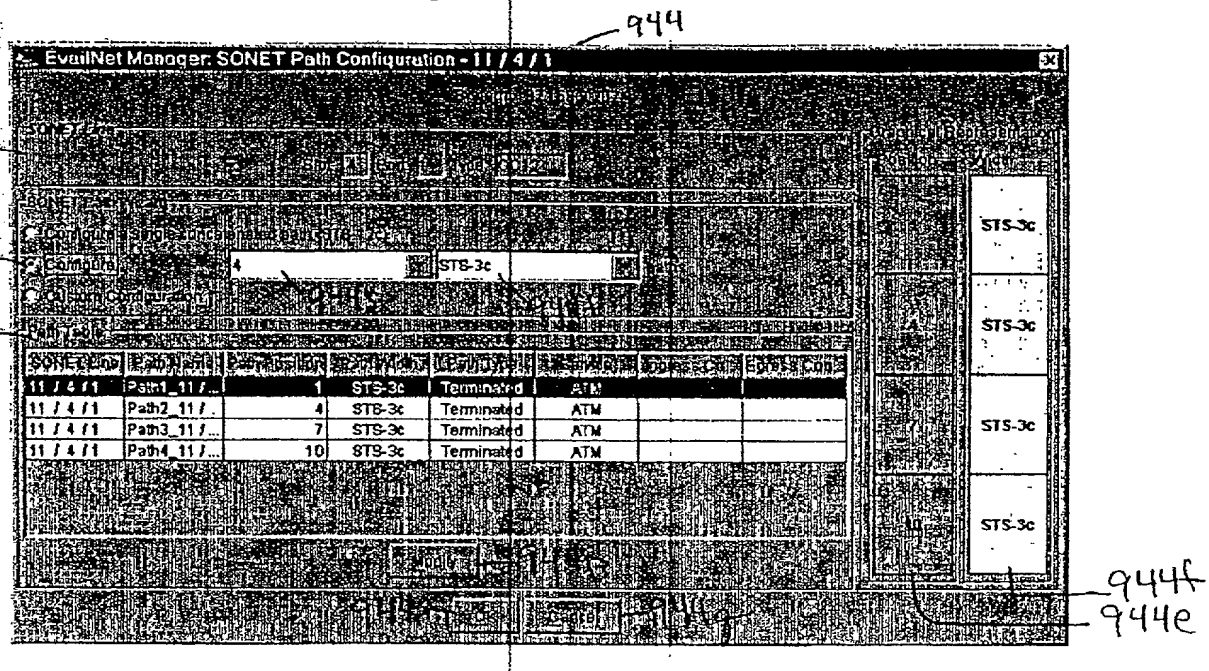

944
EvailNet Manager: SONET Path Configuration - 11 / 4 / 1
944a
944b
944c
944d
STS-3c
EvailNet Manager: SONET Path Configuration - 11 / 4 / 1
944b
STS-3c
944g
944
944e
944f

Fig. 5c

Fig. 5d


944
EvailNet Manager: SONET Path Configuration - 11 / 4 / 1
944a
944c
944g
STS-3c
STS-3c
STS-3c
STS-3c
944f
944e

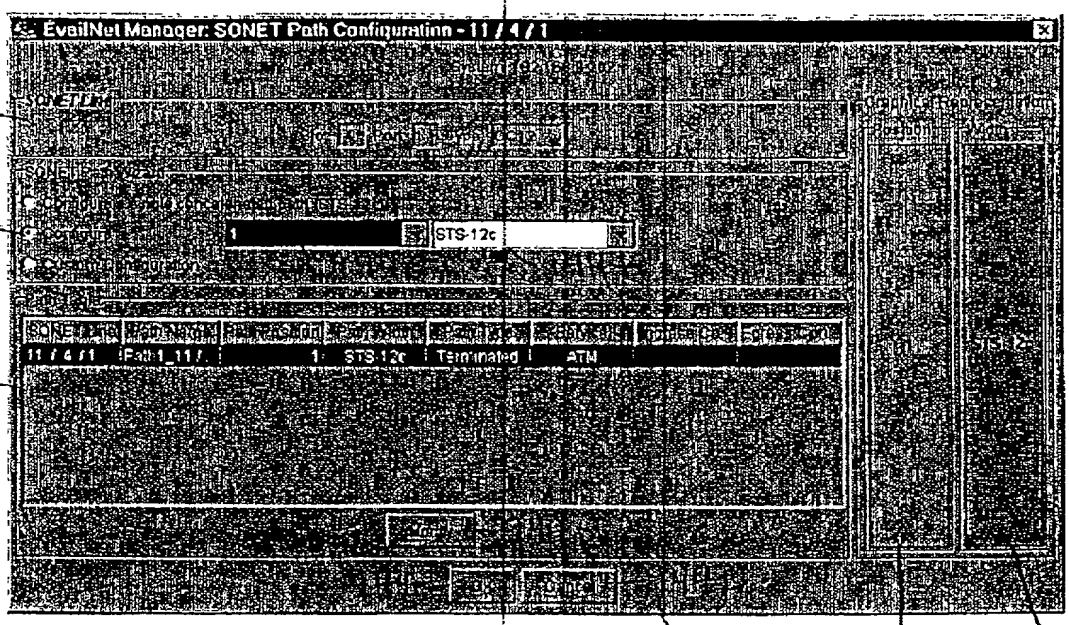
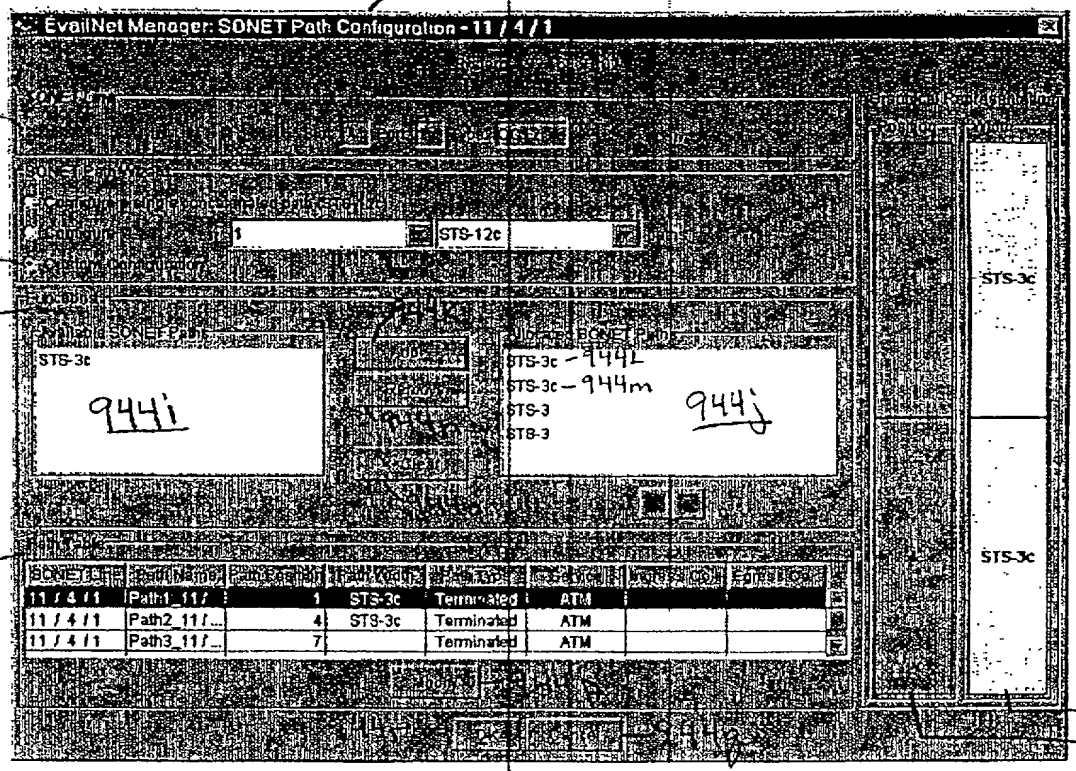

EvailNet Manager: SONET Path Configuration - 11 / 4 / 1
944a
944c
944g
944
944e
944f

Fig. 5e

Fig. 5f


944
EvalNet Manager: SONET Path Configuration - 11 / 4 / 1
944a
944d
944k
944c
STS-12c
STS-3c
944i
944j
STS-3c – 944L
STS-3c – 944m
STS-3
STS-3
944g
STS-3c
944f
944e

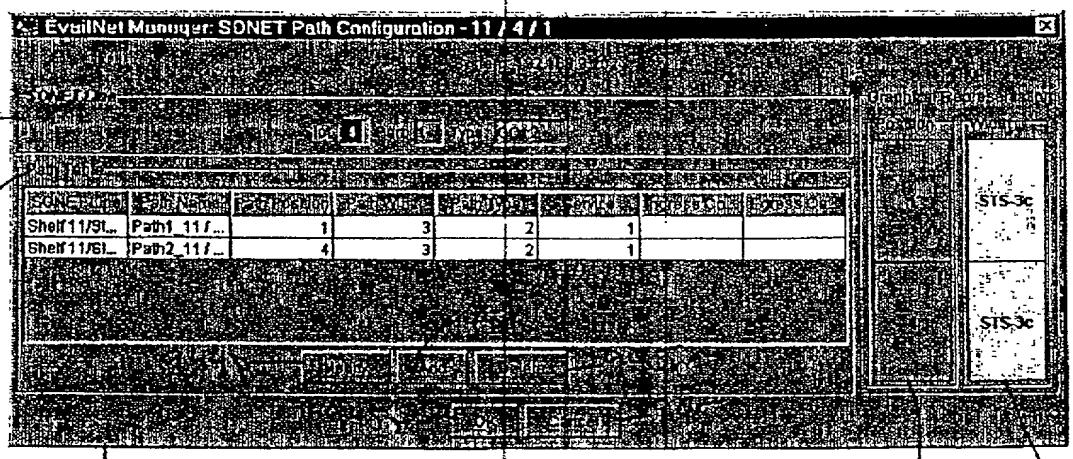

EvalNet Manager: SONET Path Configuration - 11 / 4 / 1
945a
945g
STS-3c
STS-3c
945
945e
945f

Fig. 5g

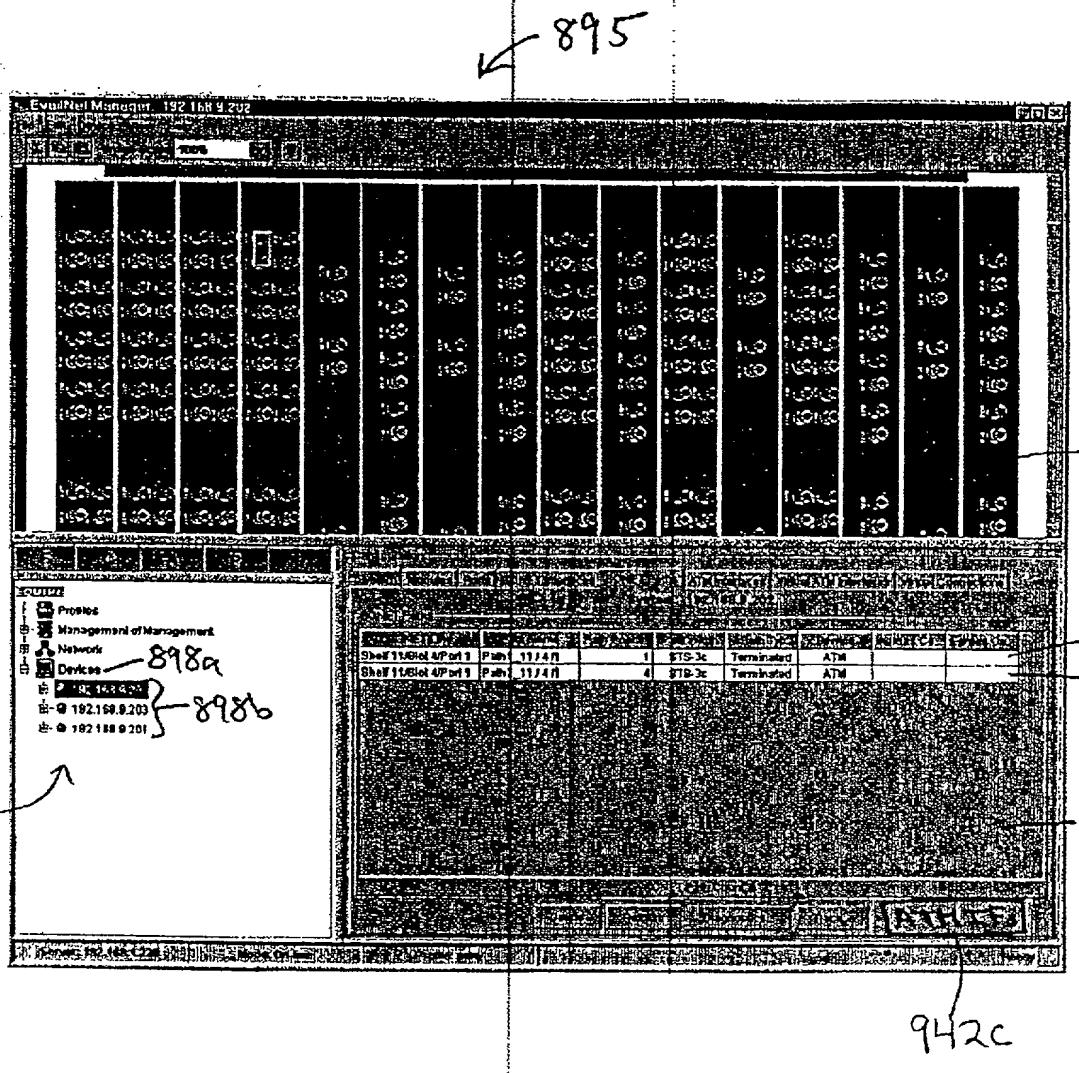

895
896a
Profiles
Management of Management
Network
Devices
898a
898b
898
942a
942b
897
942c

Fig.5h

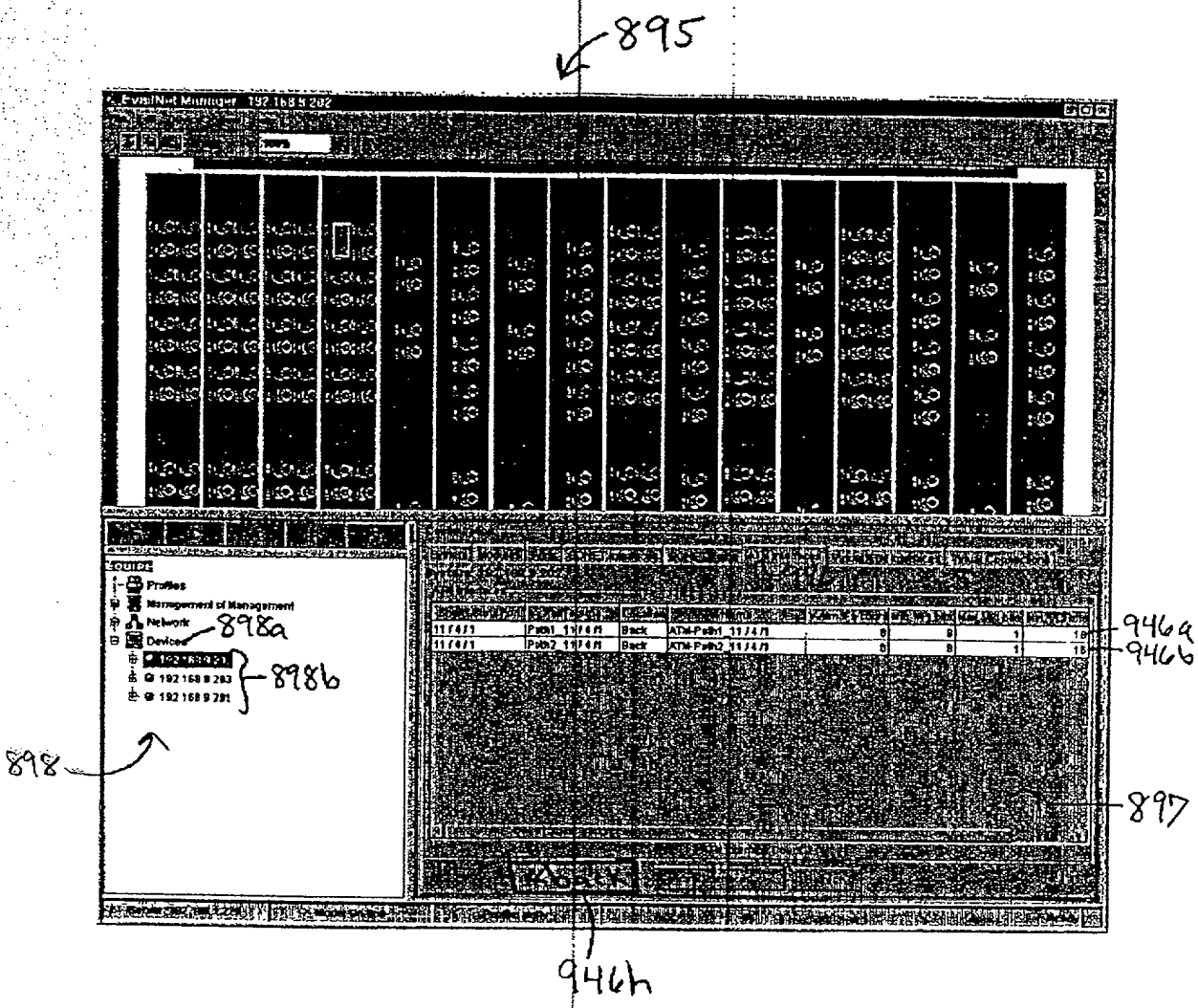


Fig. 5i

Fig. 5j

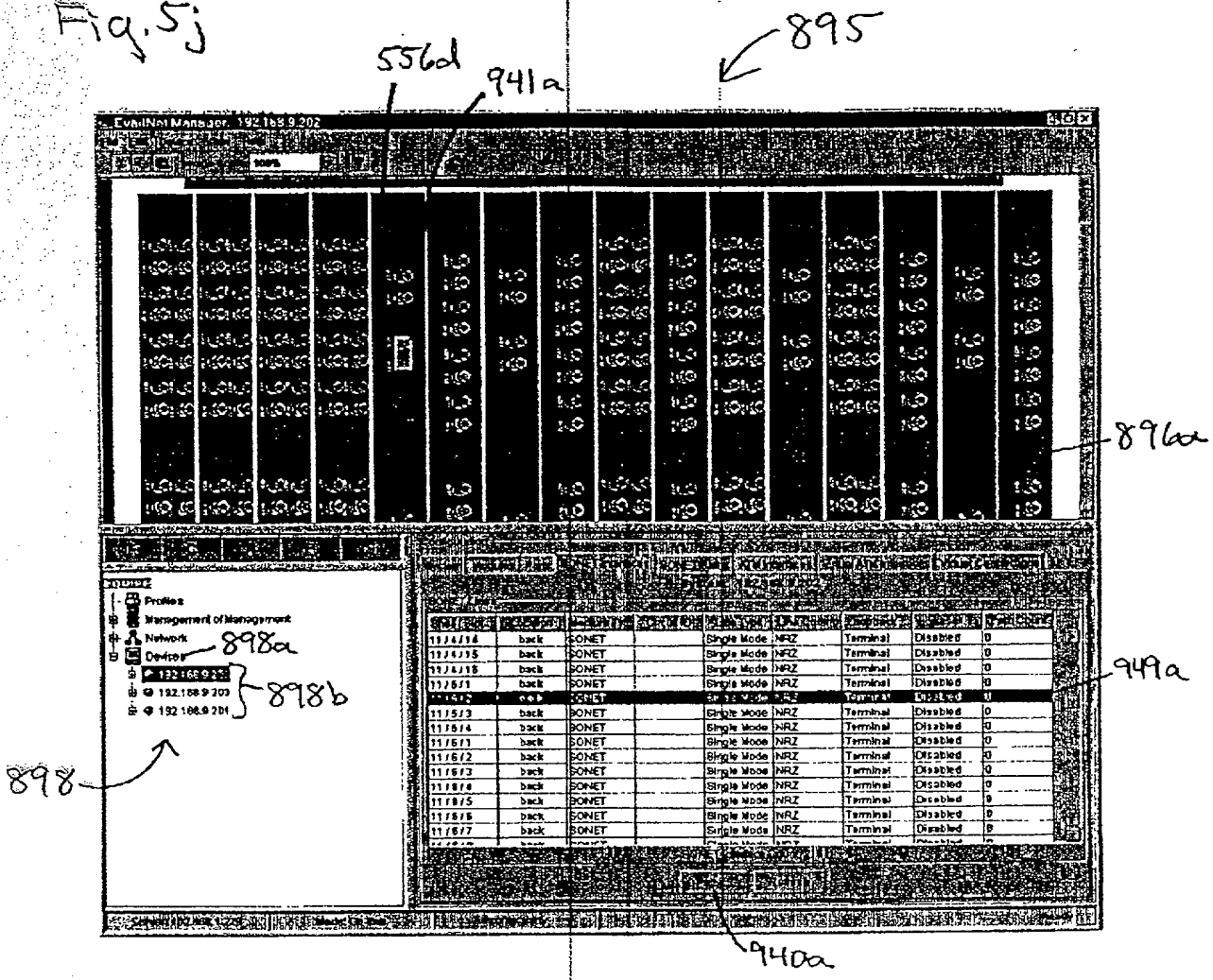

895
556d
941a
896a
898
898a
898b
899a
940a
Profiles
Management of Management
Network
Devices

Fig. 5K

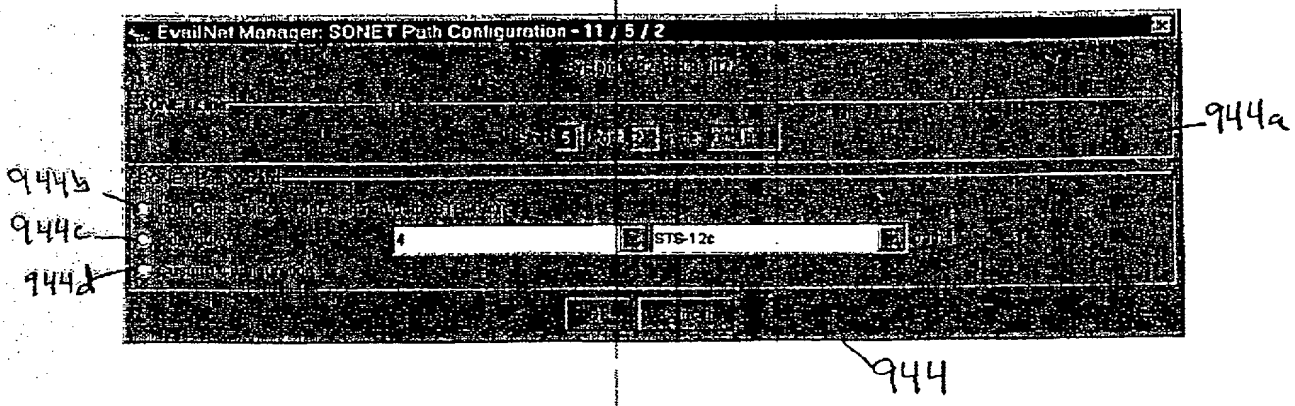
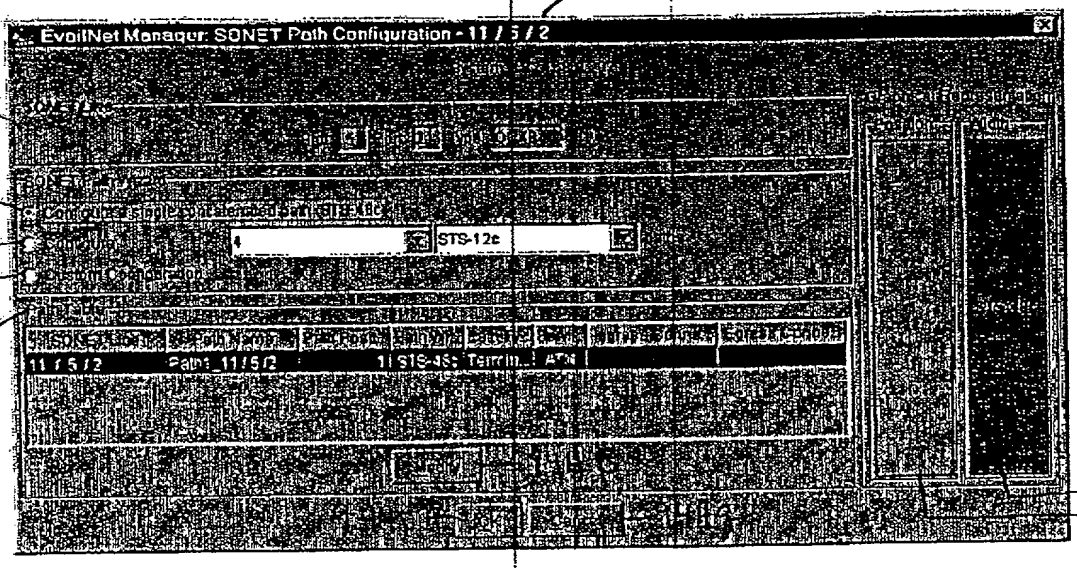

EvaliNet Manager: SONET Path Configuration - 11 / 5 / 2
944a
944b
944c
944d
STS-12c
944

Fig. 5l


944
EvailNet Manager: SONET Path Configuration - 11 / 5 / 2
944a
944b
944c
944d
944g
STS-12c
11 / 5 / 2
Path1_11/5/2
944f
944e

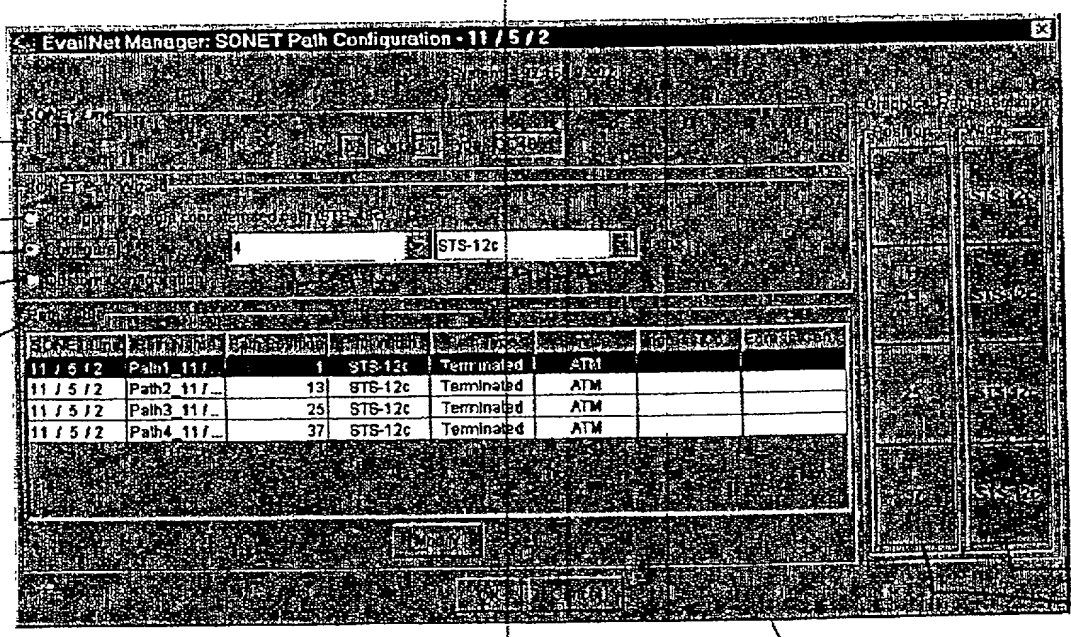

EvailNet Manager: SONET Path Configuration - 11 / 5 / 2
944a
944b
944c
944d
944g
STS-12c
11 / 5 / 2 Path1_11/... 1 STS-12c Terminated ATM
11 / 5 / 2 Path2_11/... 13 STS-12c Terminated ATM
11 / 5 / 2 Path3_11/... 25 STS-12c Terminated ATM
11 / 5 / 2 Path4_11/... 37 STS-12c Terminated ATM
944f
944e
944

Fig. 5m

Fig. 5n

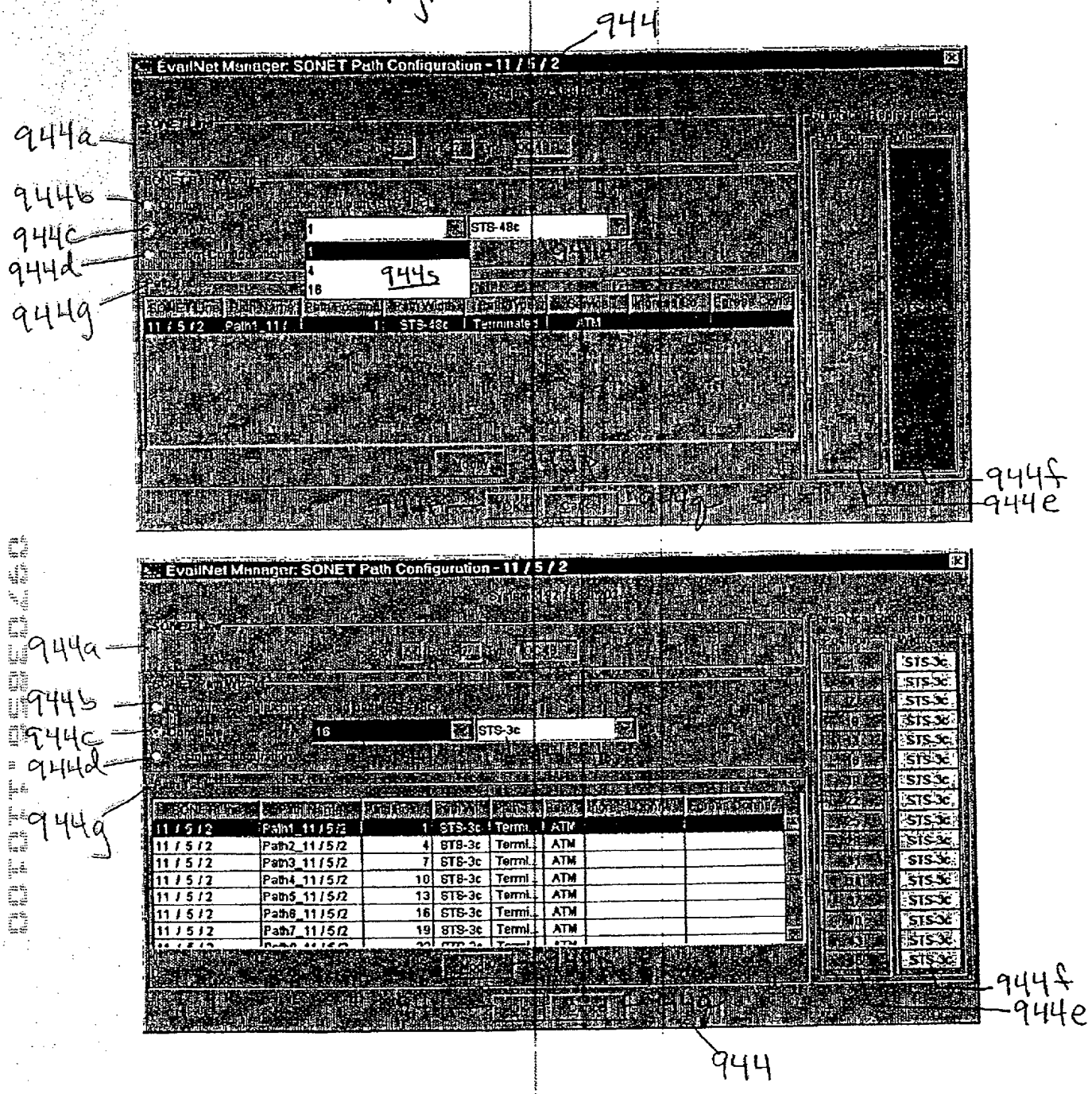

944
EvailNet Manager: SONET Path Configuration - 11 / 5 / 2
944a
944b
944c
944d
944g
STB-48c
1
4
16
944h
944f
944e
EvailNet Manager: SONET Path Configuration - 11 / 5 / 2
944a
944b
944c
944d
944g
16
STS-3c
11 / 5 / 2 Path1_11 / 5 / 2 1 STS-3c Termi... ATM
11 / 5 / 2 Path2_11 / 5 / 2 4 STS-3c Termi... ATM
11 / 5 / 2 Path3_11 / 5 / 2 7 STS-3c Termi... ATM
11 / 5 / 2 Path4_11 / 5 / 2 10 STS-3c Termi... ATM
11 / 5 / 2 Path5_11 / 5 / 2 13 STS-3c Termi... ATM
11 / 5 / 2 Path6_11 / 5 / 2 16 STS-3c Termi... ATM
11 / 5 / 2 Path7_11 / 5 / 2 19 STS-3c Termi... ATM
944f
944e
944

Fig. 5o

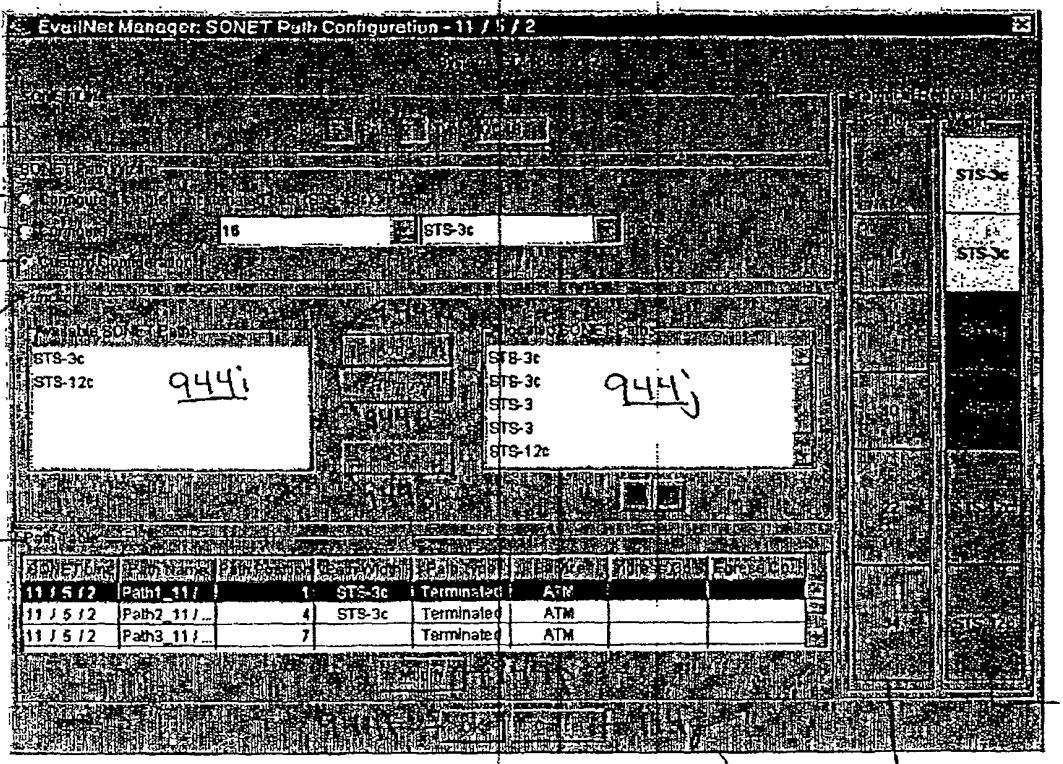


Fig. 5P

SUBSTITUTE SHEET (RULE 26)

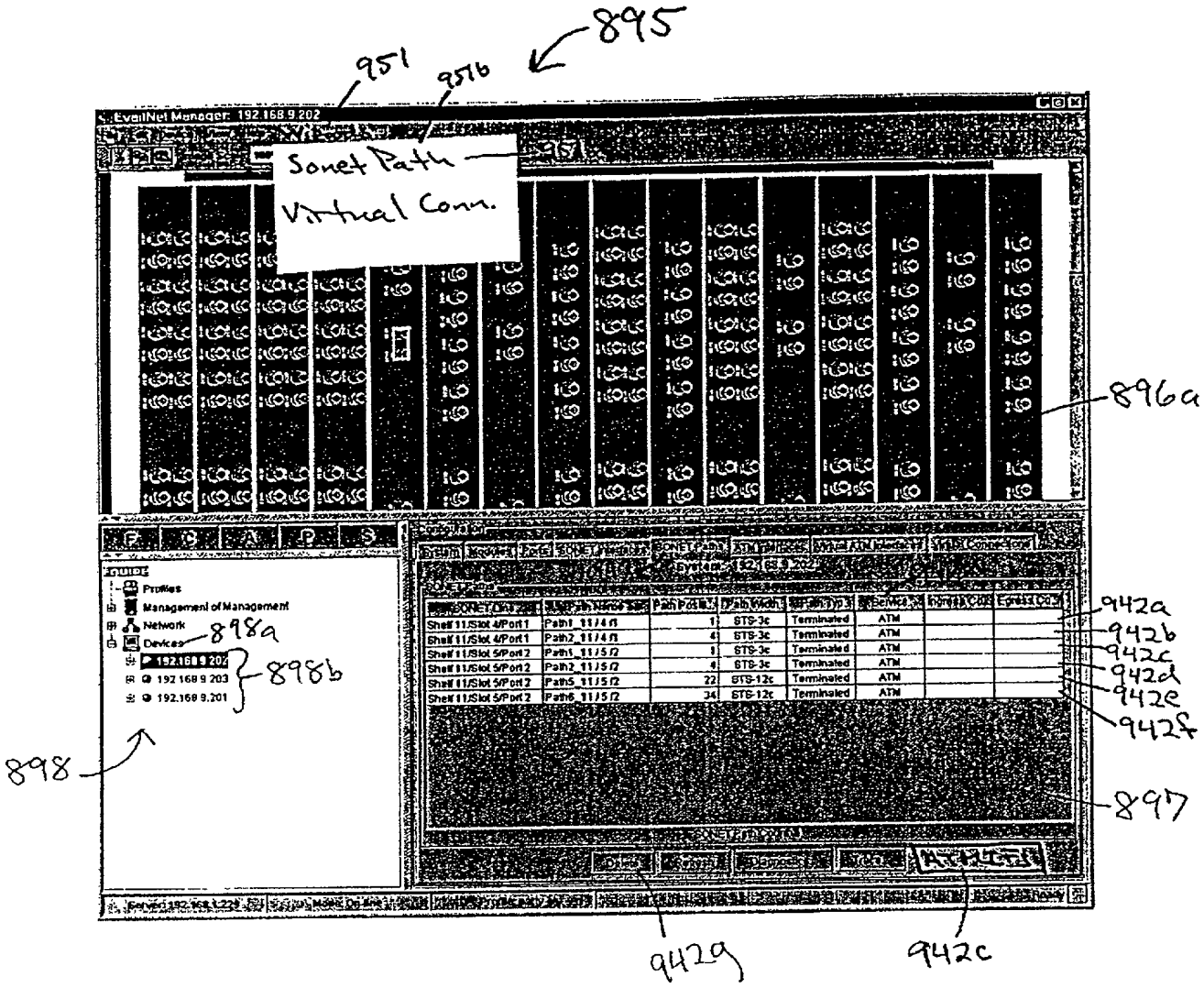

895
951
956
Sonet Path
Virtual Conn.
896a
898a
898b
898
942a
942b
942c
942d
942e
942f
897
942g
942c

Fig. 5g

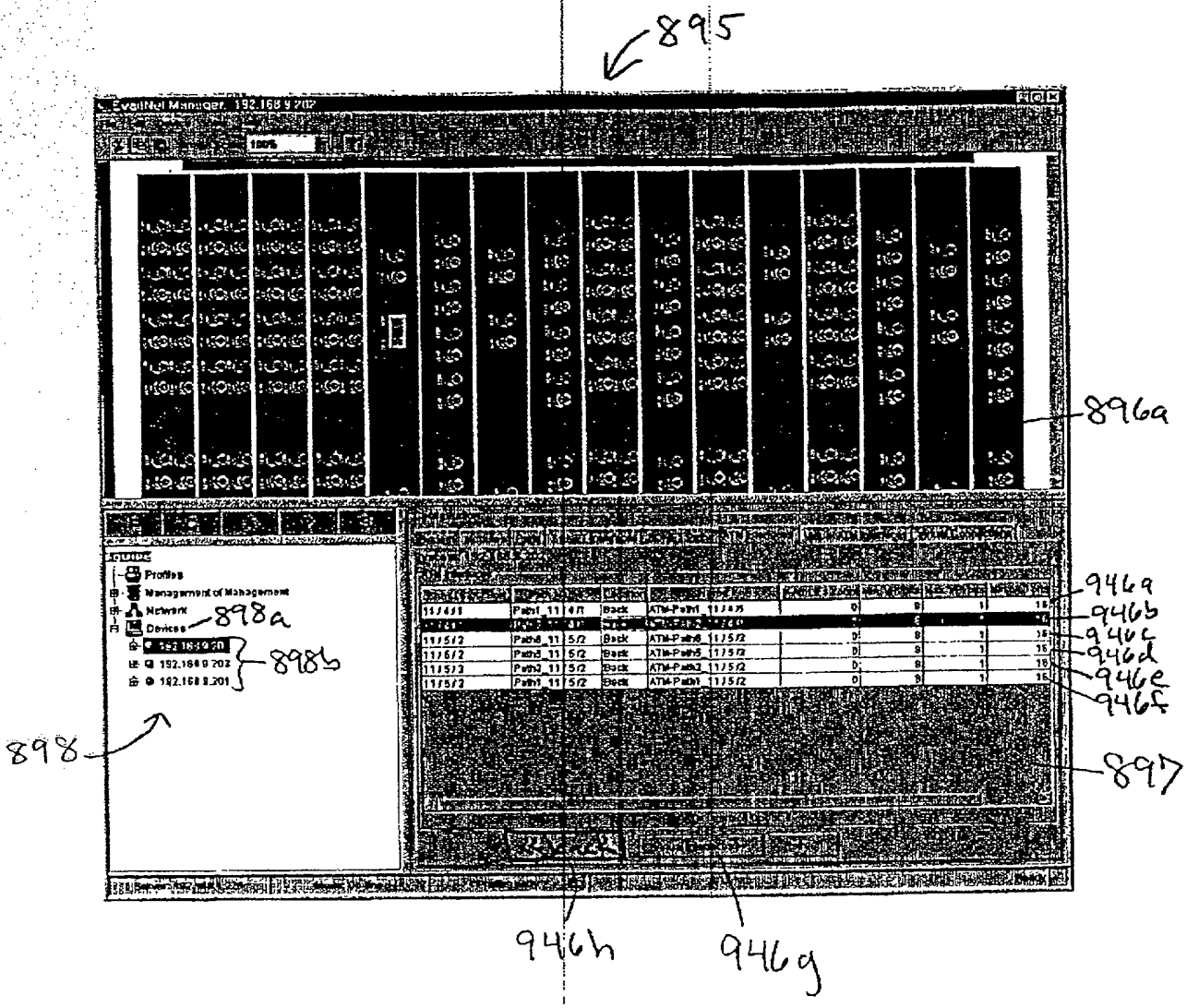

895
896a
898a
898b
898
946a
946b
946c
946d
946e
946f
897
946h
946g

Fig. 5F

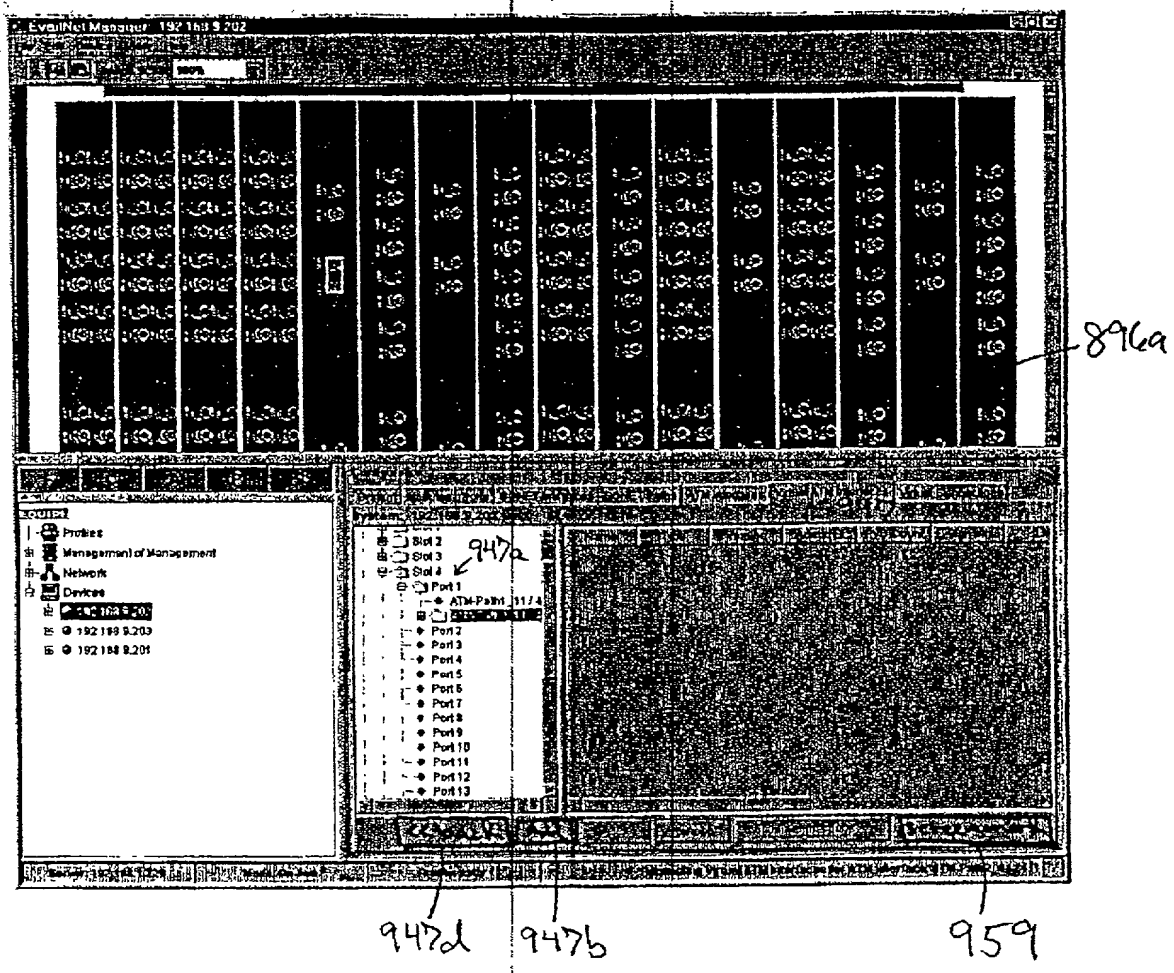
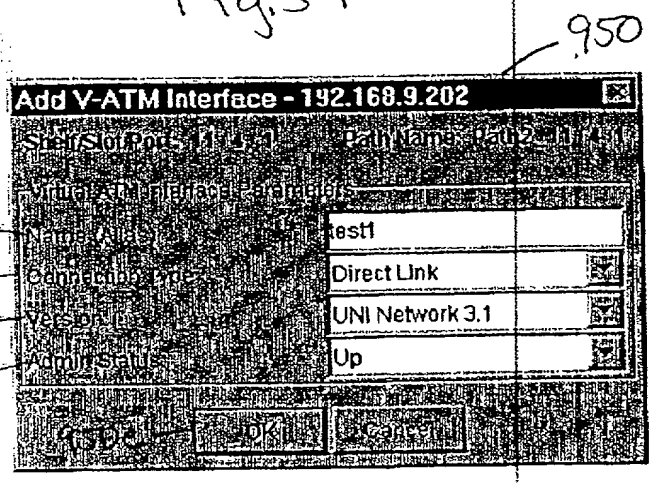


Fig. 5s

attorney docket 4569

Fig. 5t


950
Add V-ATM Interface - 192.168.9.202
950d
test1
950a
Direct Link
950b
UNI Network 3.1
950c
Up
OK
Cancel

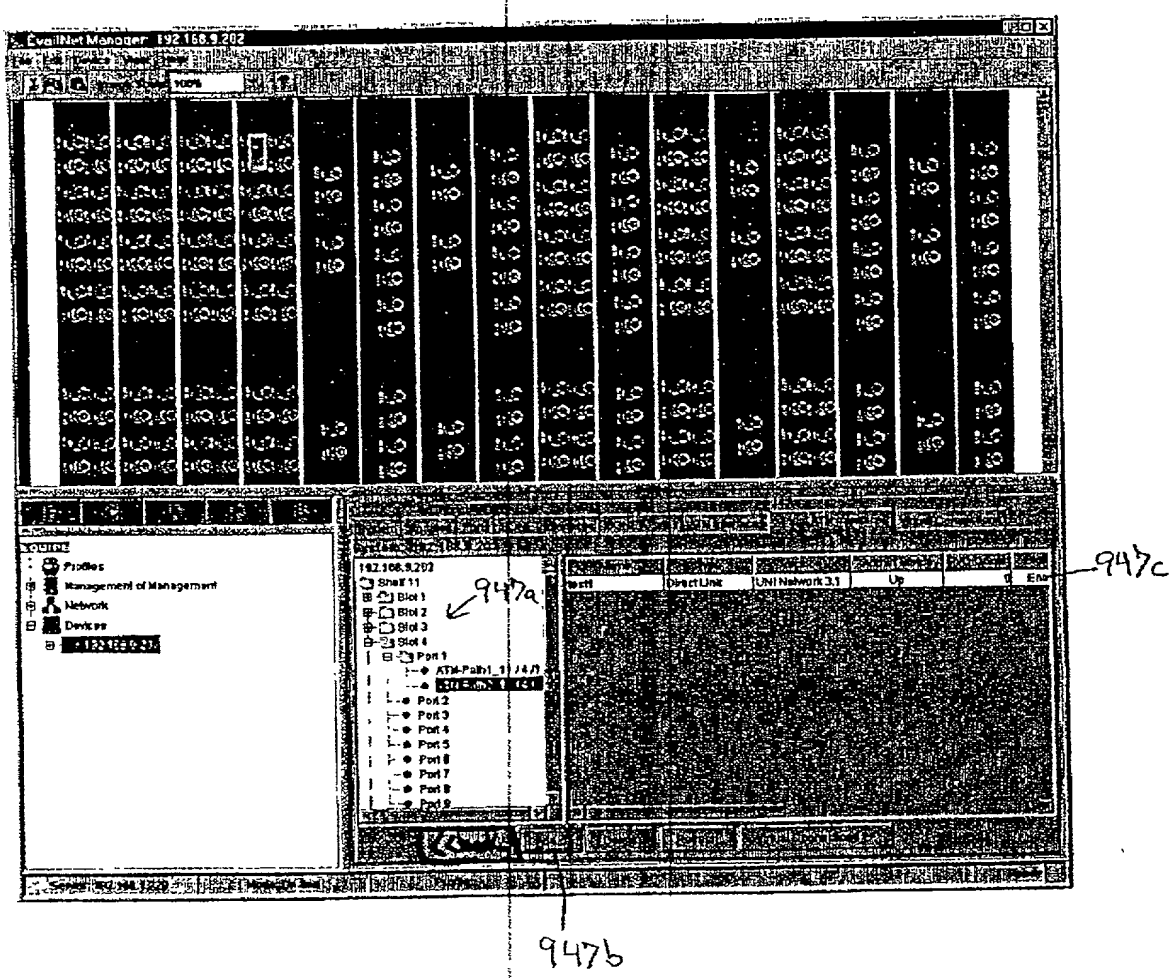

947a
947c
947b

Fig. 5u

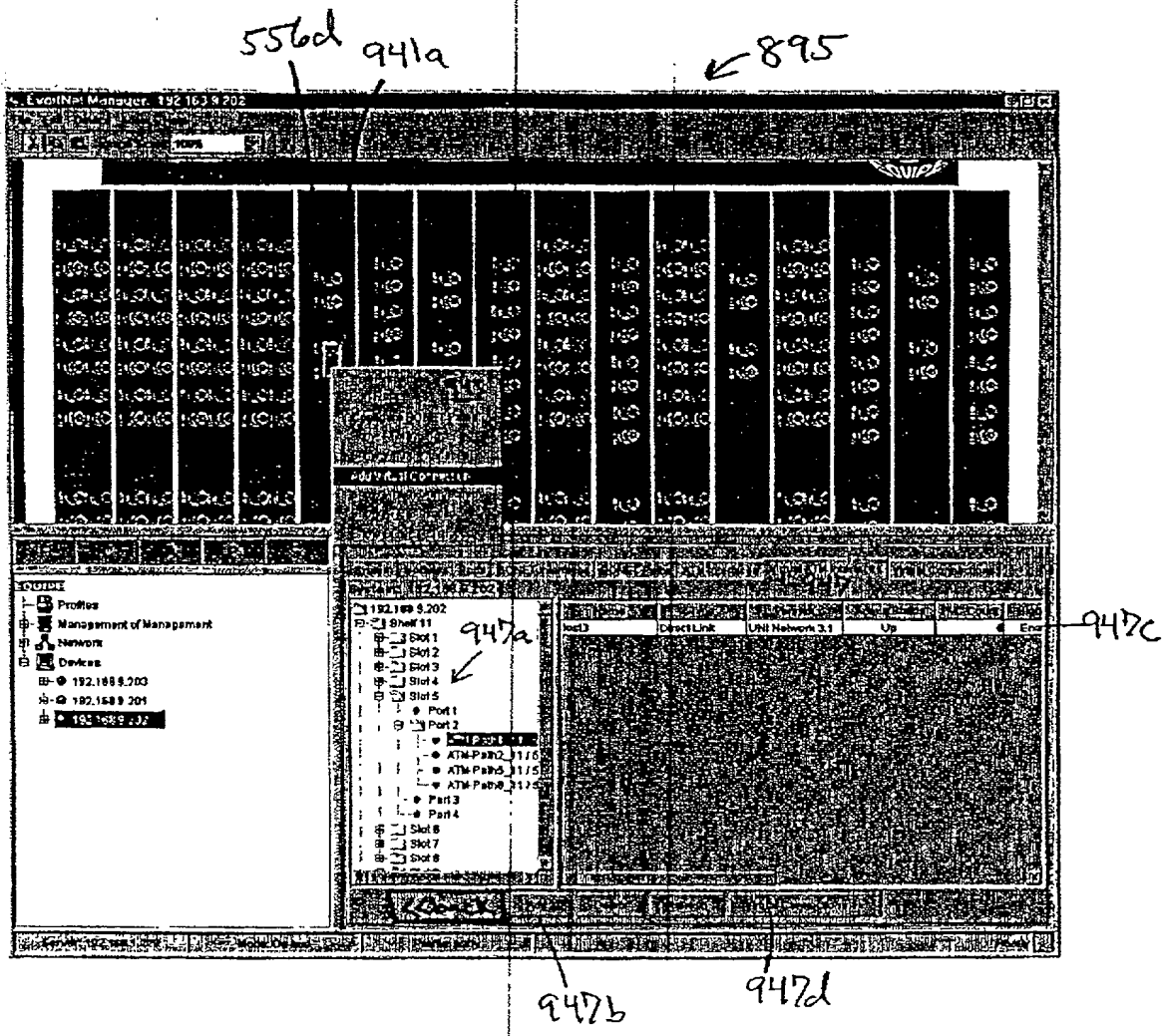


Fig. 5v

952

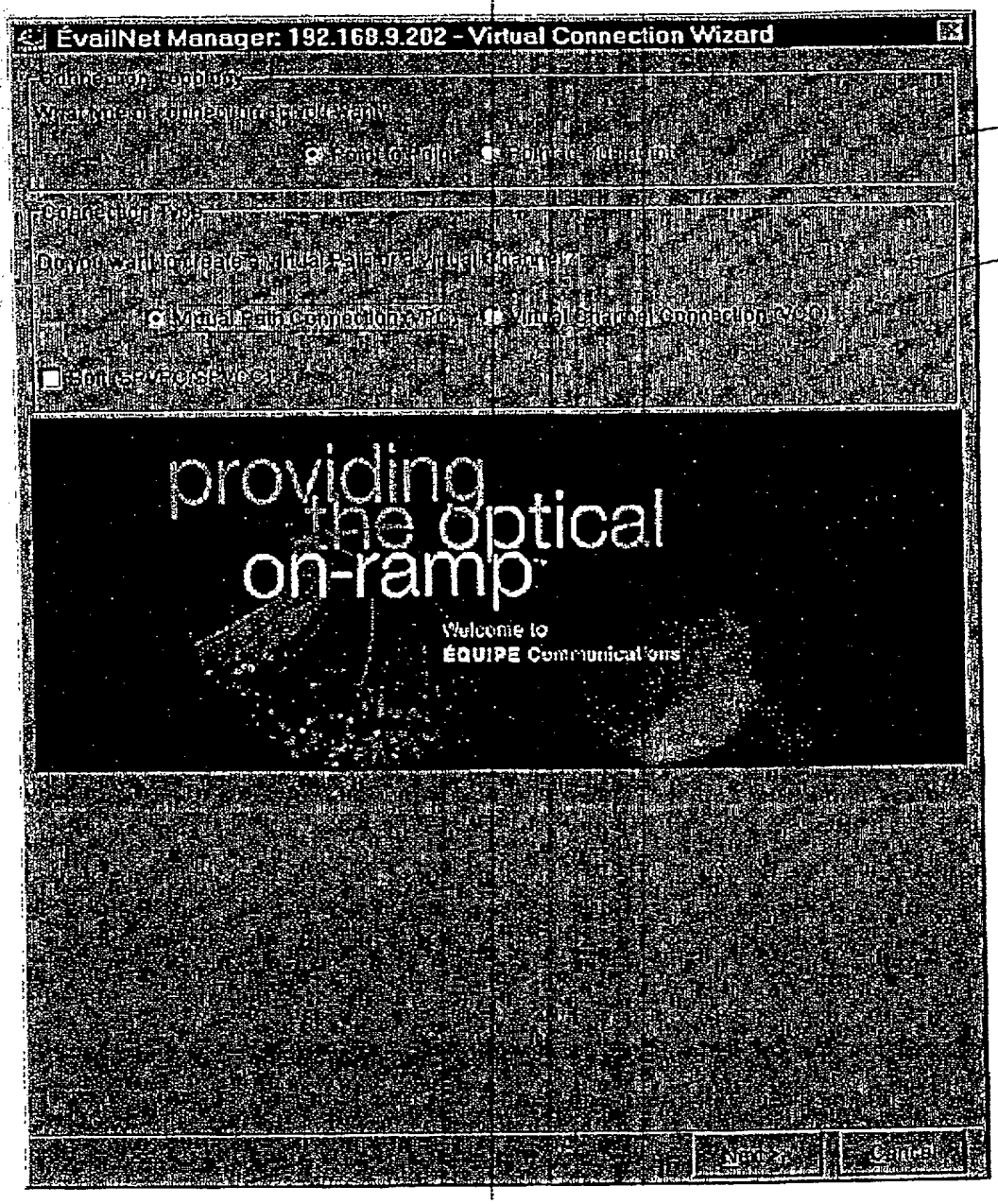

EvailNet Manager: 192.168.9.202 - Virtual Connection Wizard
952a
952b
providing the optical on-ramp
Welcome to
EQUIPE Communications

Fig. 5w

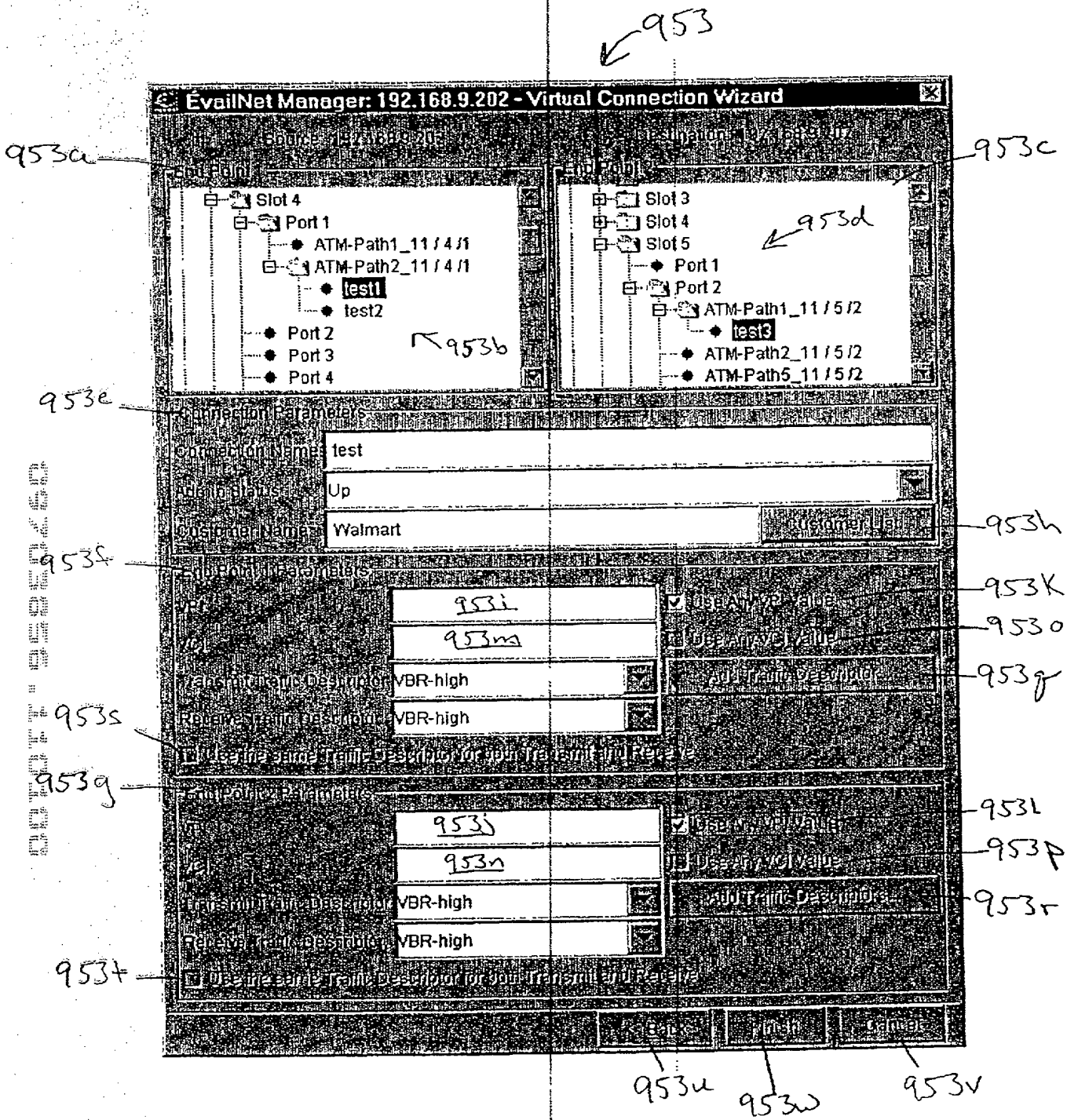

953
EvailNet Manager: 192.168.9.202 - Virtual Connection Wizard
953a
953c
Slot 4
Port 1
ATM-Path1_11 / 4 /1
ATM-Path2_11 / 4 /1
test1
test2
Port 2
Port 3
Port 4
953b
Slot 3
Slot 4
Slot 5
953d
Port 1
Port 2
ATM-Path1_11 / 5 /2
test3
ATM-Path2_11 / 5 /2
ATM-Path5_11 / 5 /2
953e
Connection Parameters
Connection Name: test
Admin Status: Up
Customer Name: Walmart
Customer List
953h
953f
Sub Port Parameters
953i
953m
953K
953o
Transmit Traffic Descriptor: VBR-high
Receive Traffic Descriptor: VBR-high
Add Traffic Descriptor
953g
953s
Use the same Traffic Descriptor for both Transmit and Receive
953q
Sub Port Parameters
953j
953n
953l
953p
Transmit Traffic Descriptor: VBR-high
Receive Traffic Descriptor: VBR-high
Add Traffic Descriptor
953r
953t
Use the same Traffic Descriptor for both Transmit and Receive
Finish
Cancel
953u
953w
953v

Fig. 5x

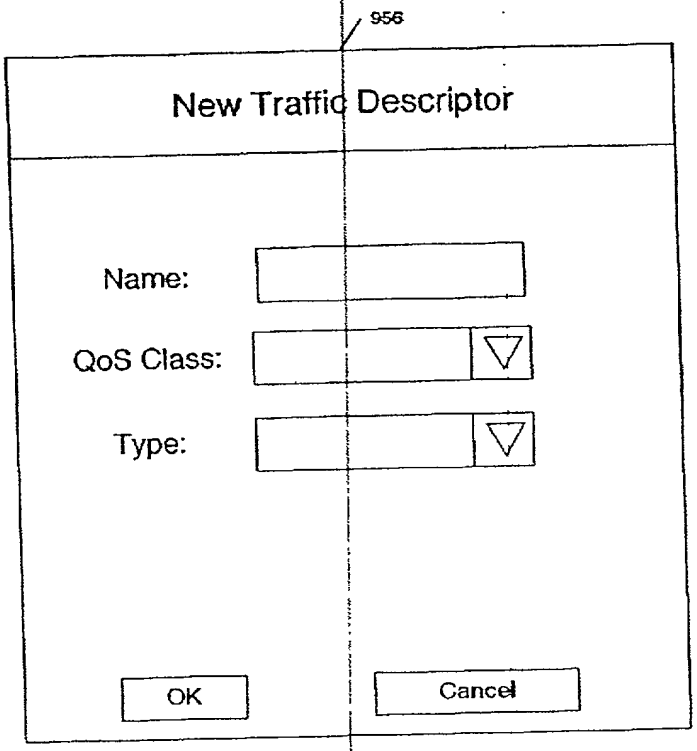


Fig. 5Y

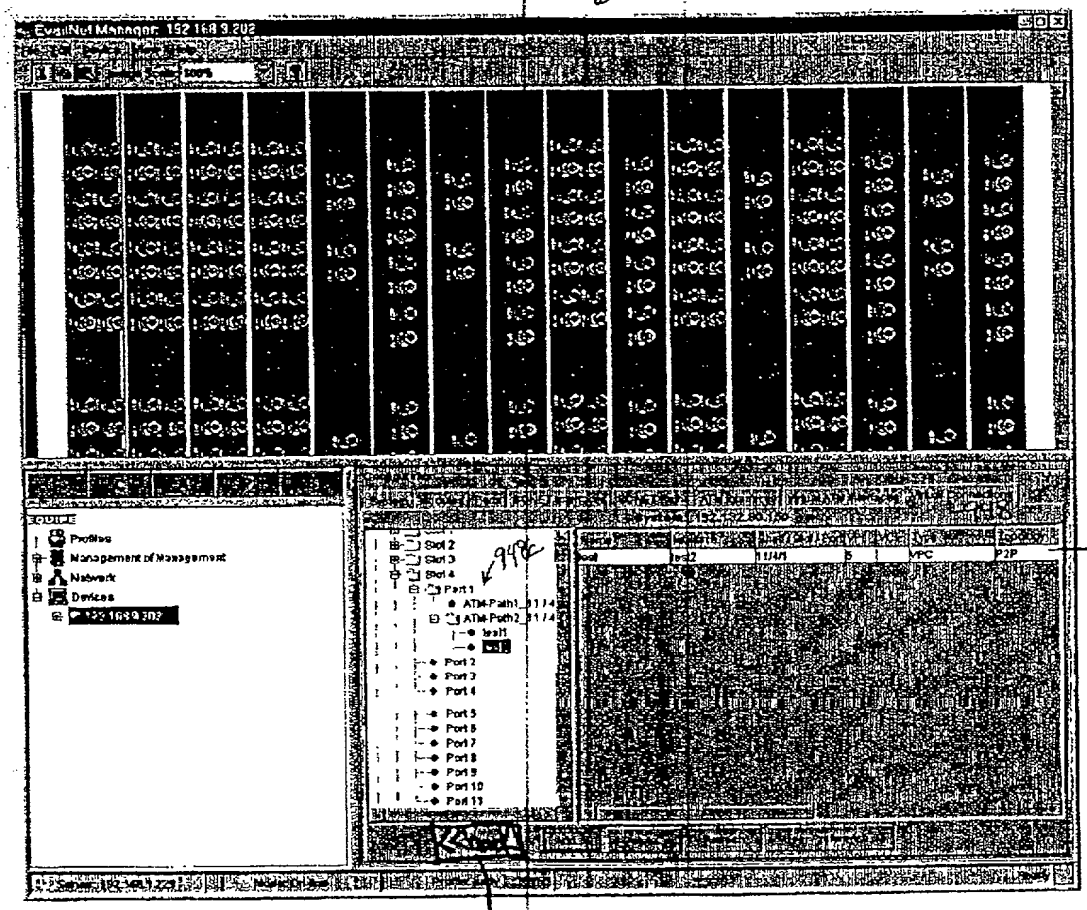


Fig. 57

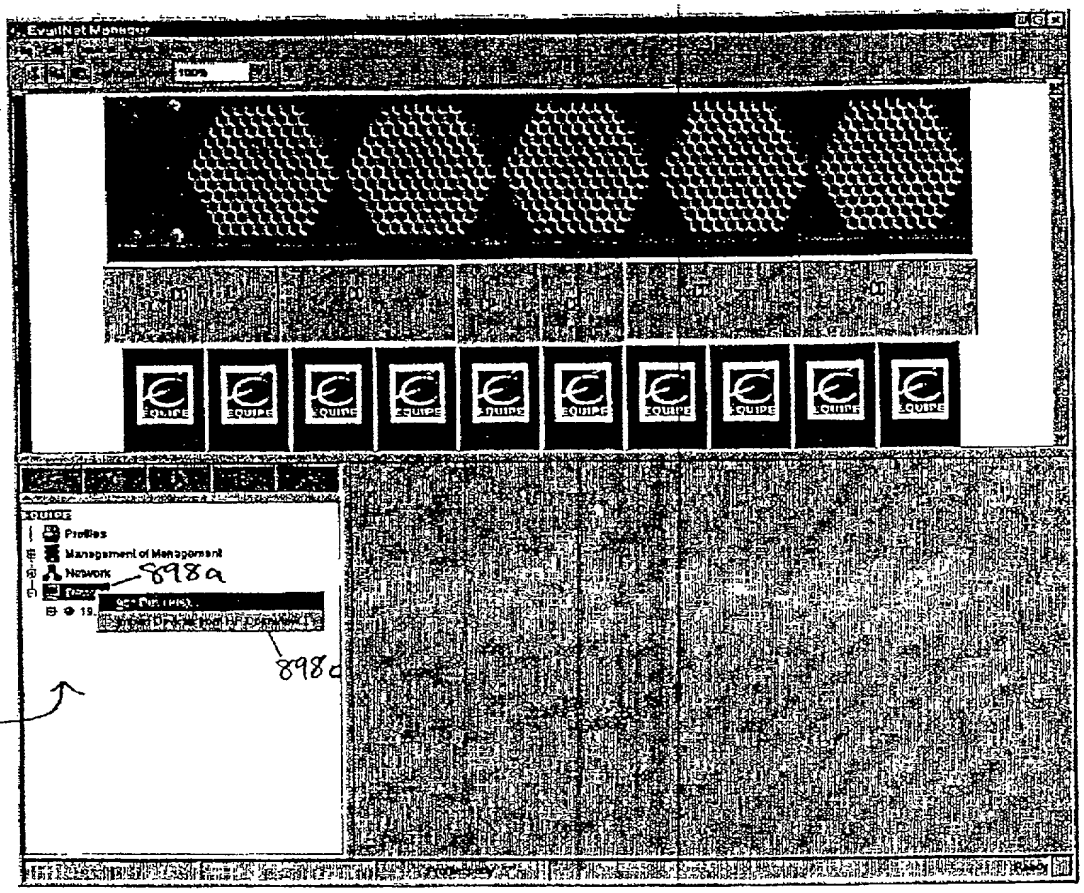

895
EquiNet Manager
EQUIPE
Profiles
Management of Management
Network
898a
898c
898

Fig. 6a

Fig. 6b

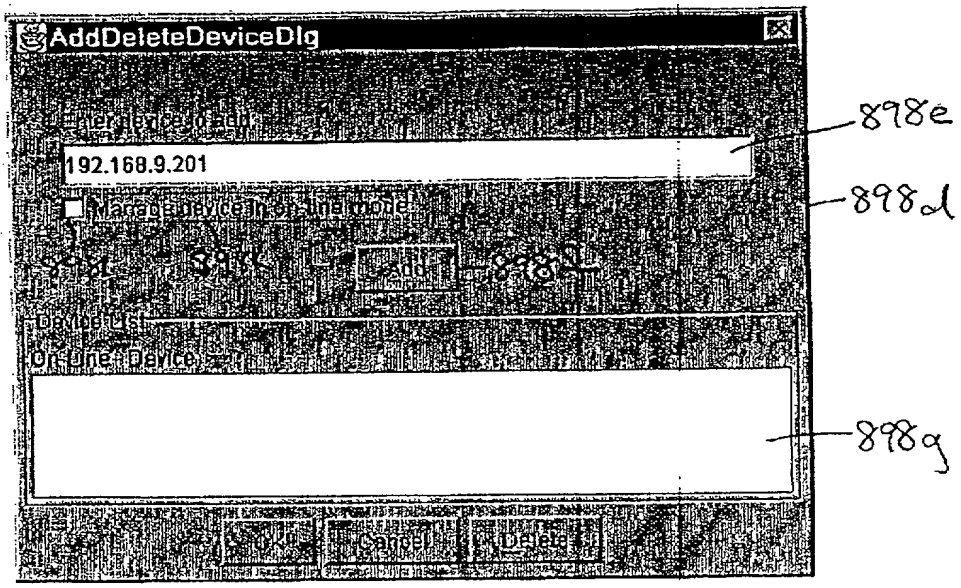
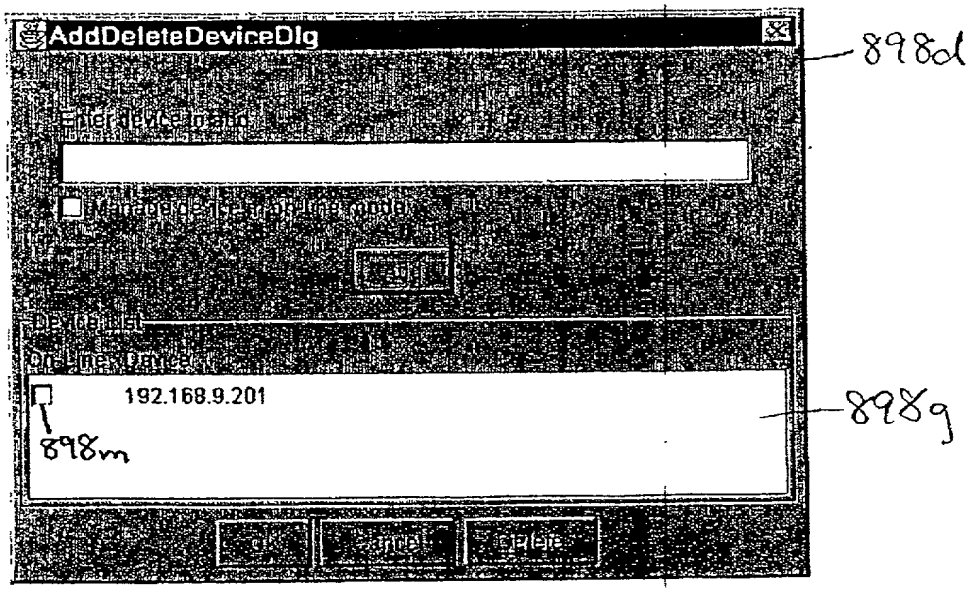

AddDeleteDeviceDlg
Enter device to add
192.168.9.201
Manage device in online mode
Add
Device List
On-Line Device
OK
Cancel
Delete
898e
898d
898g

Fig. 6c


AddDeleteDeviceDlg
Enter device to add
Manage device in online mode
Add
Device List
On-Line Device
192.168.9.201
898m
OK
Cancel
Delete
898d
898g

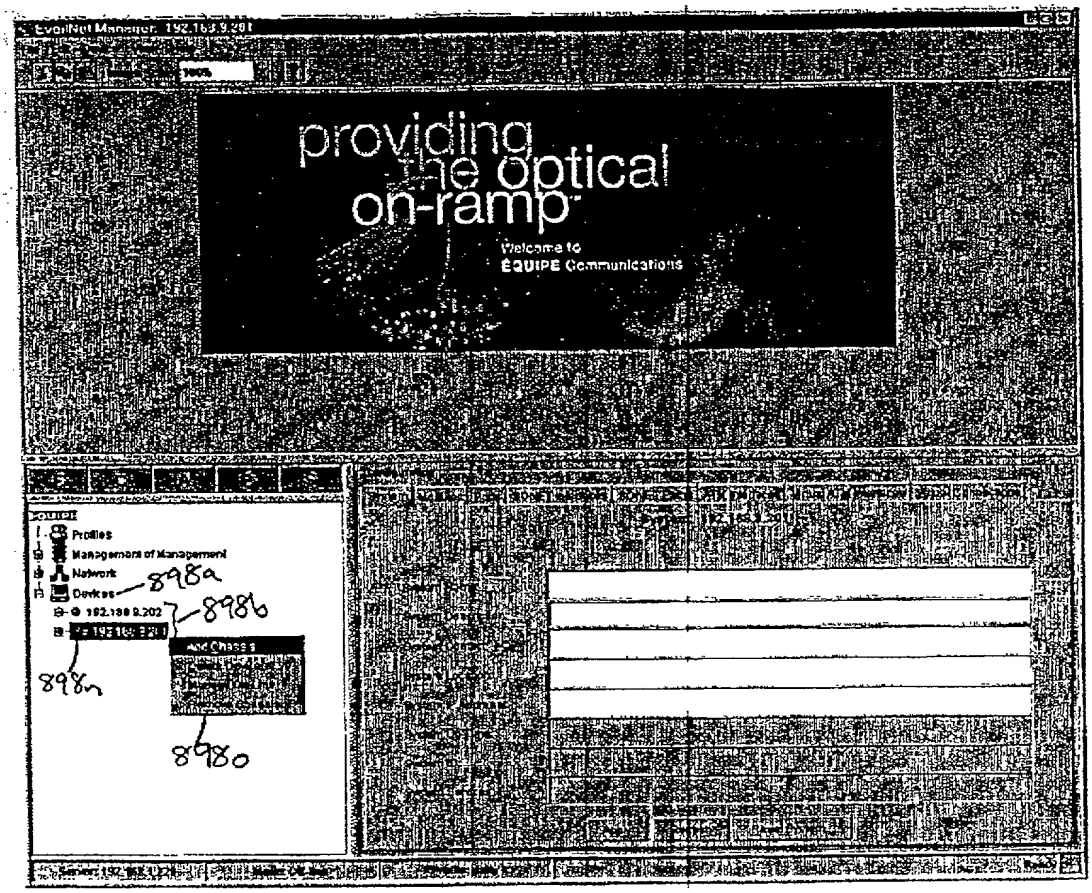

895
providing the optical on-ramp
Welcome to
EQUIPE Communications
Profiles
Management of Management
Network
Devices
898a
8986
898n
8980

Fig. 6d

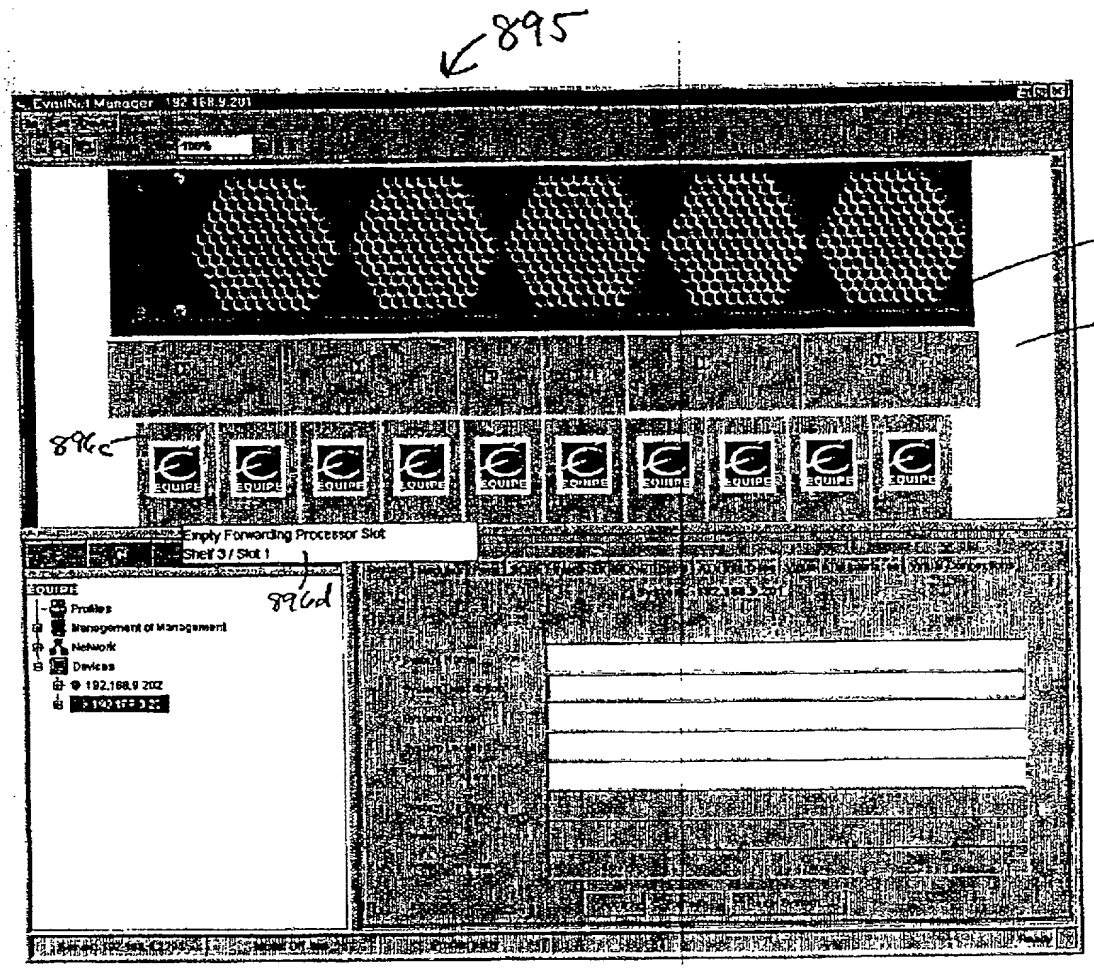


Fig. 6e

DOCKET A L A R M

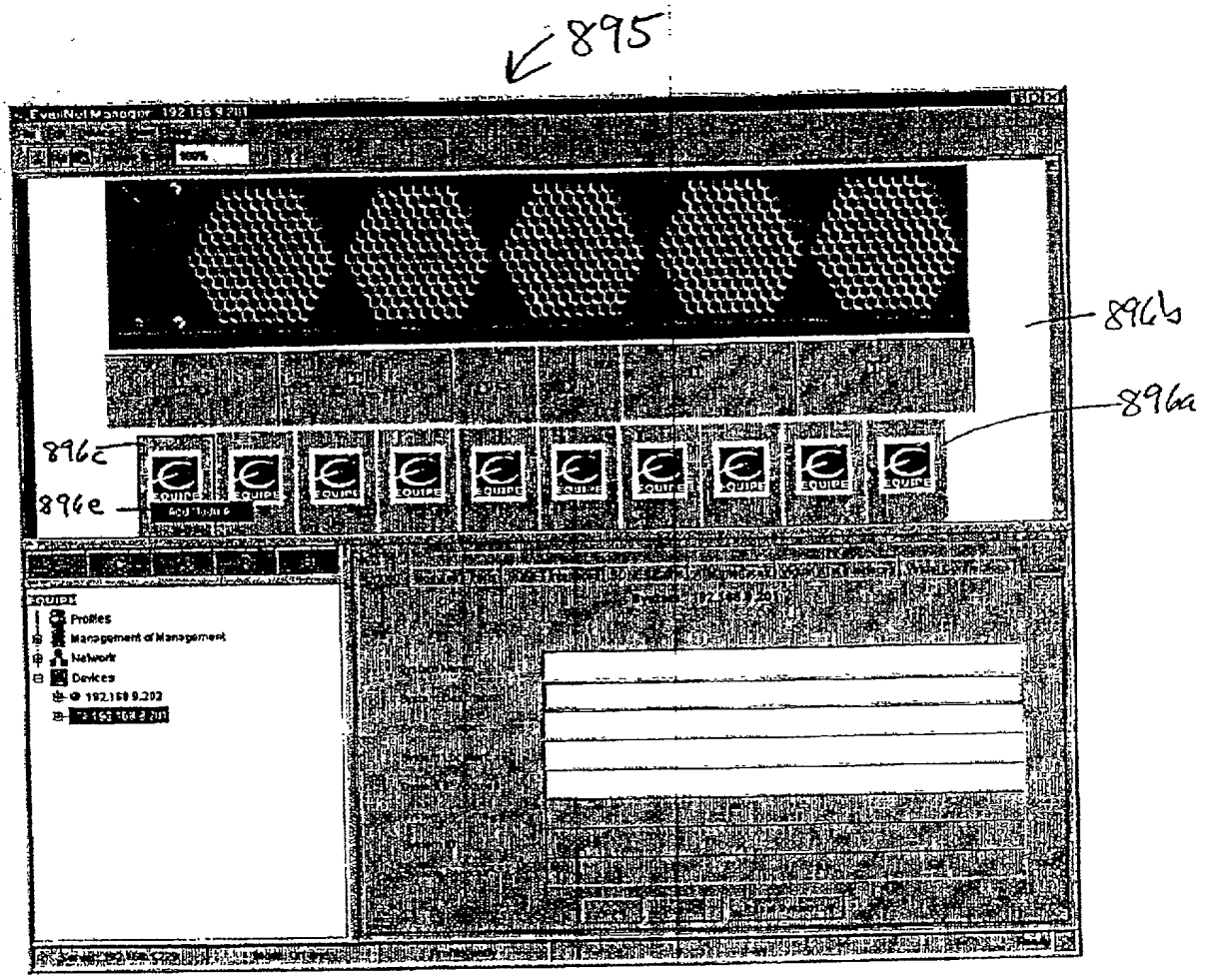

895
896b
896a
896c
896e
Profiles
Management of Management
Network
Devices

Fig. 6f

895

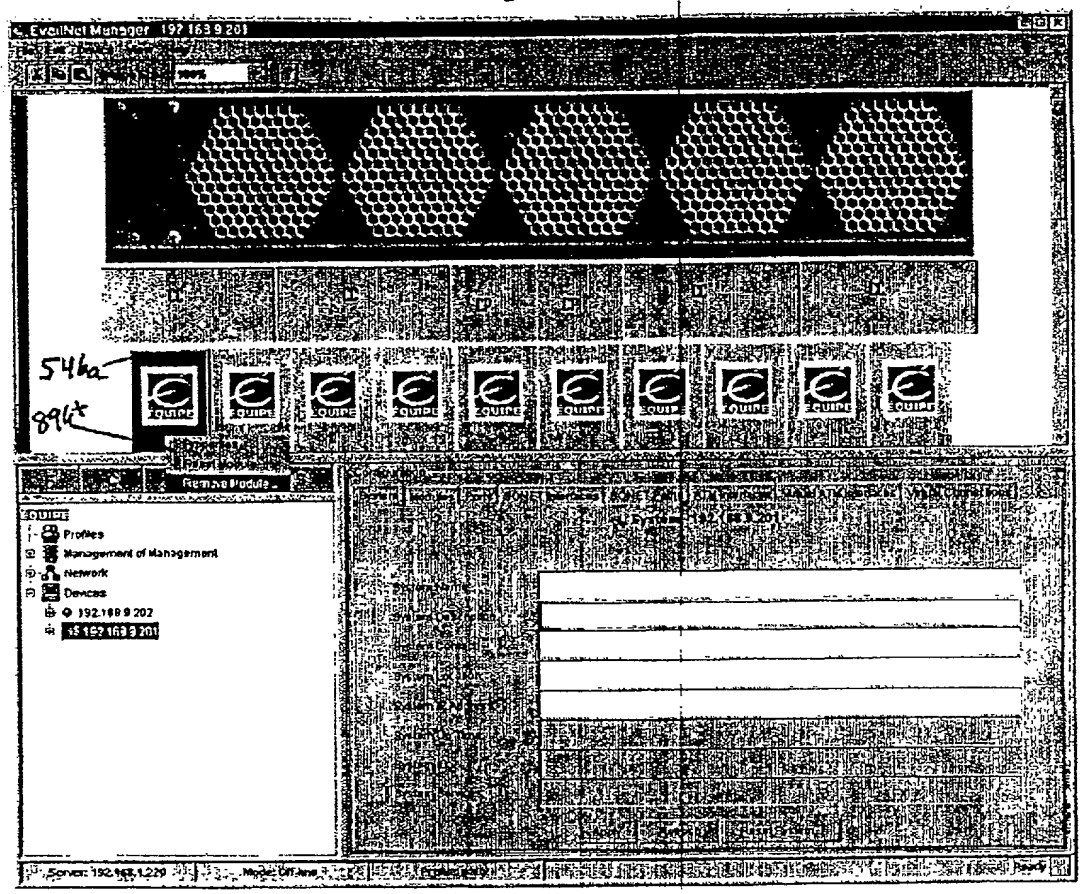

546a
896
Profiles
Management of Management
Network
Devices

Fig. 6g

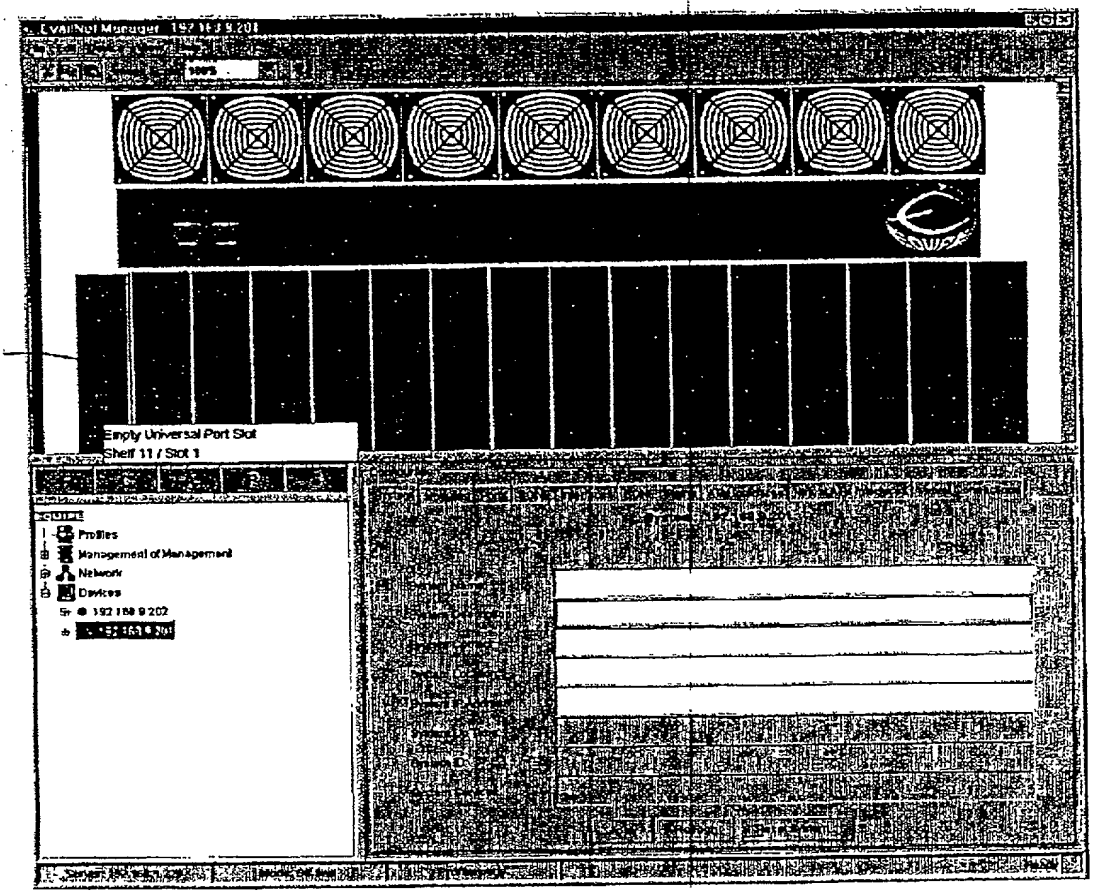

895
896
Empty Universal Port Slot
Shelf 11 / Slot 1
Profiles
Management of Management
Network
Devices

Fig. 6h

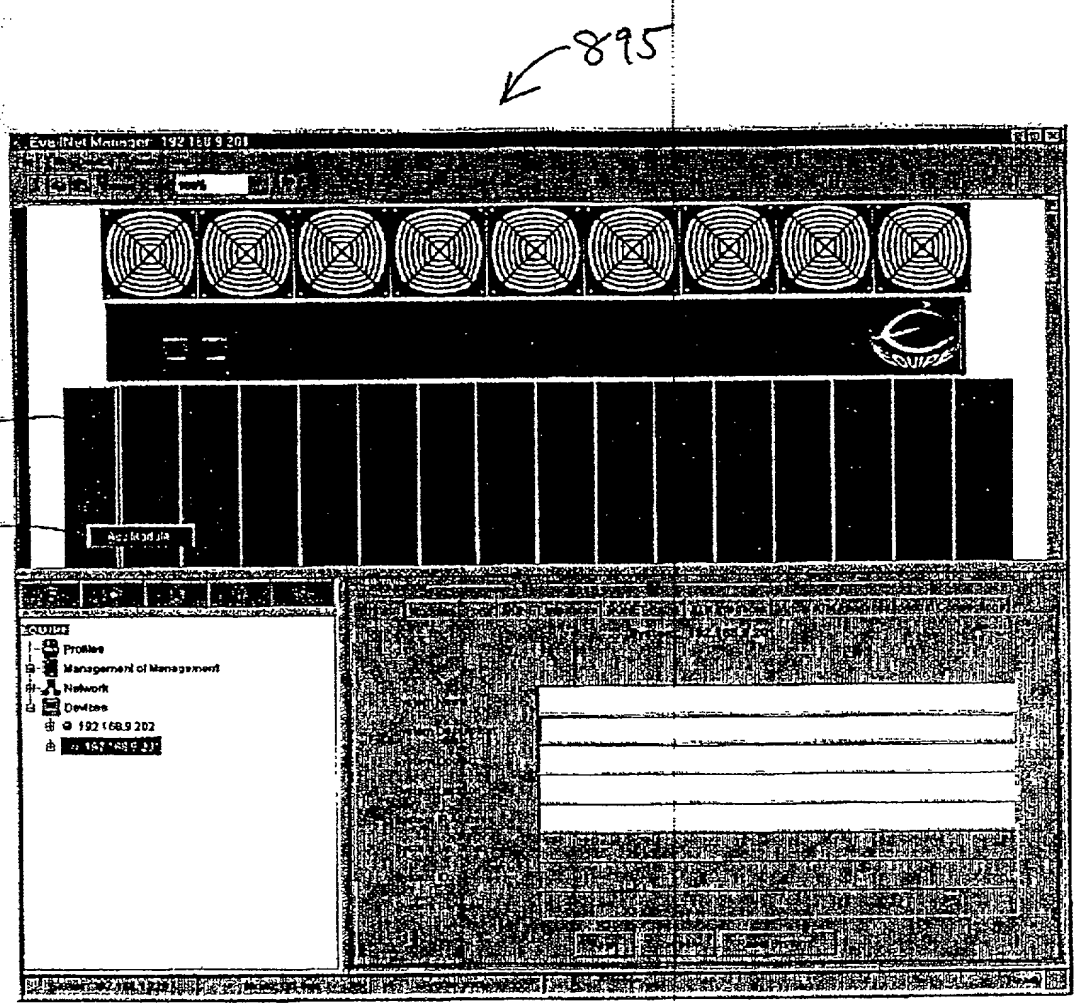
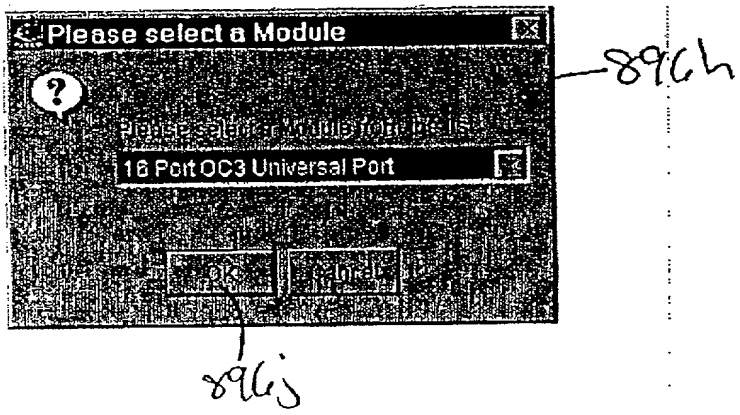

895
896f
896g
Profiles
Management of Management
Network
Devices

Fig. 6i

Fig. 6J


Please select a Module
16 Port OC3 Universal Port
OK
Cancel
896h
896j

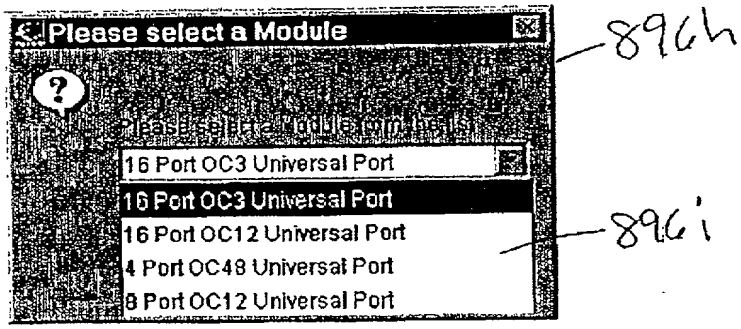

Please select a Module
16 Port OC3 Universal Port
16 Port OC3 Universal Port
16 Port OC12 Universal Port
4 Port OC48 Universal Port
8 Port OC12 Universal Port
896h
896i

Fig. 6K

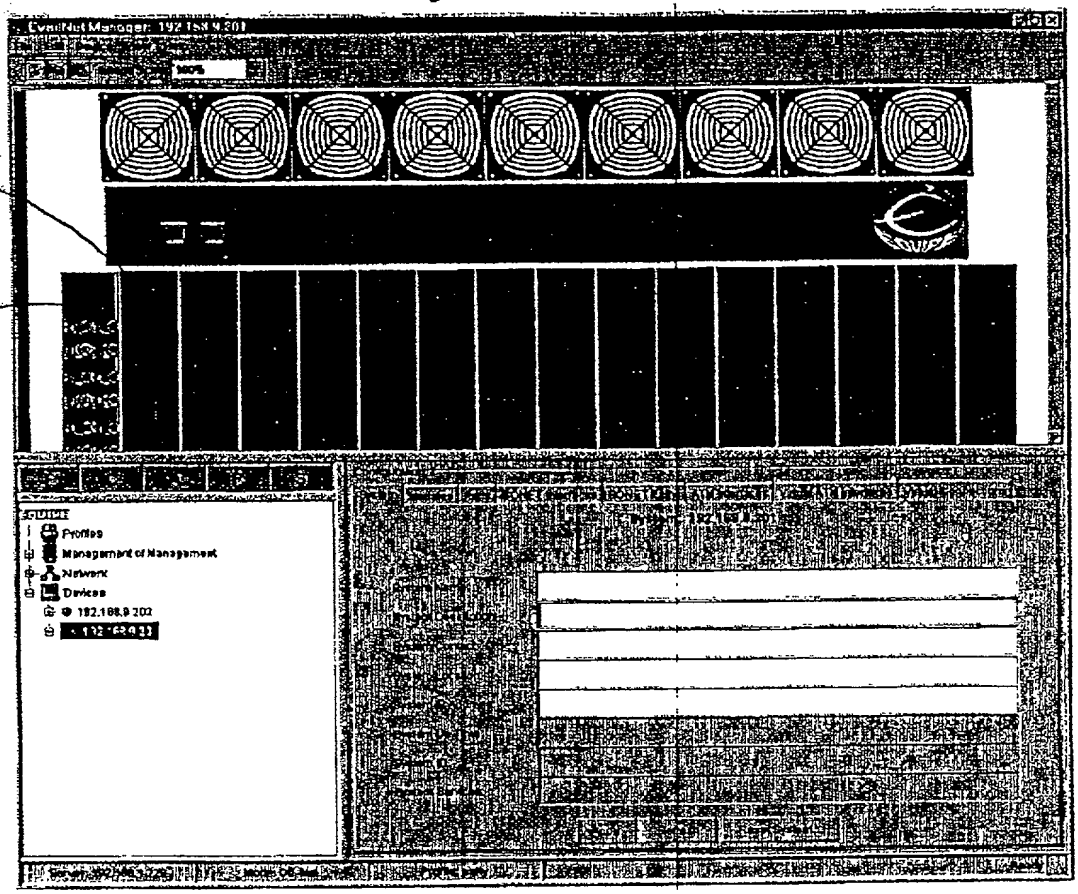

895
5561
556h
Profiles
Management of Management
Network
Devices

Fig. 6L

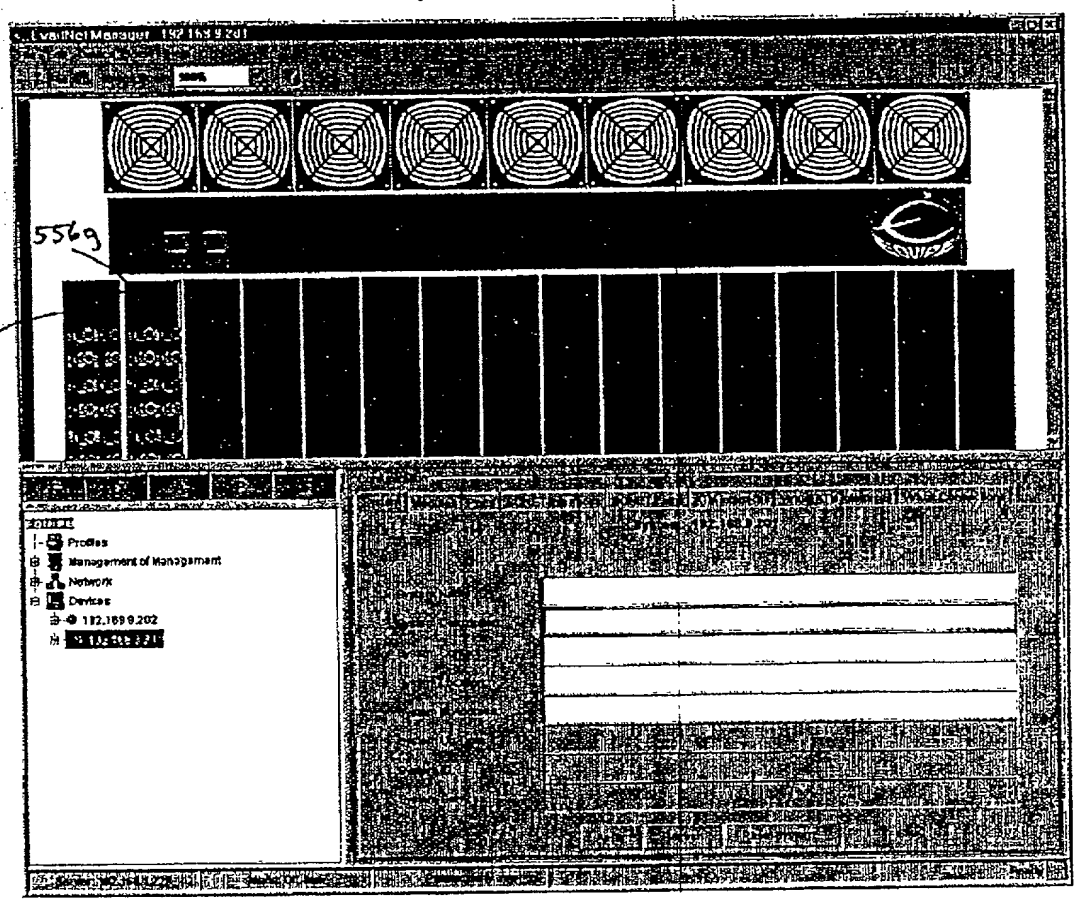

895
556g
556h
Profiles
Management of Management
Network
Devices

Fig. 6m

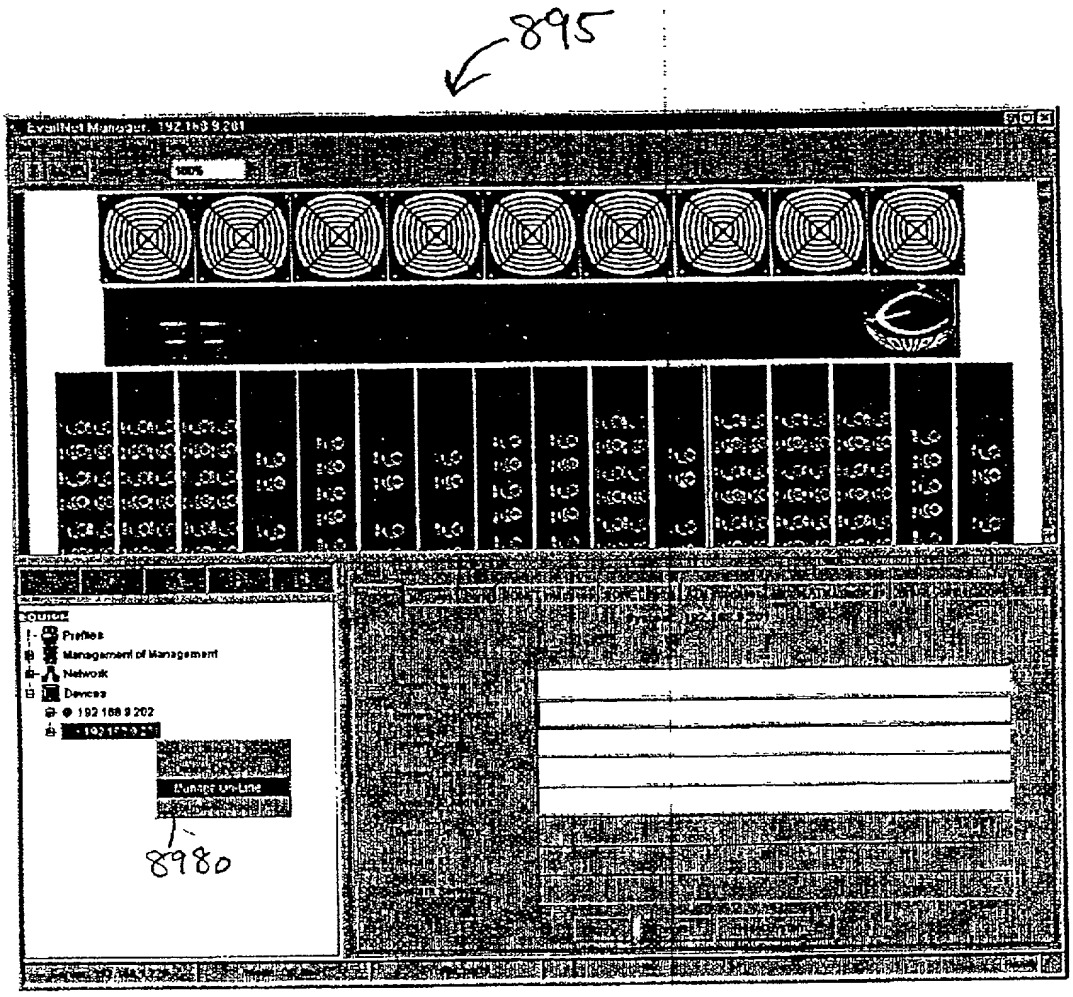

895
Profiles
Management of Management
Network
Devices
8980

Fig. 6n

895

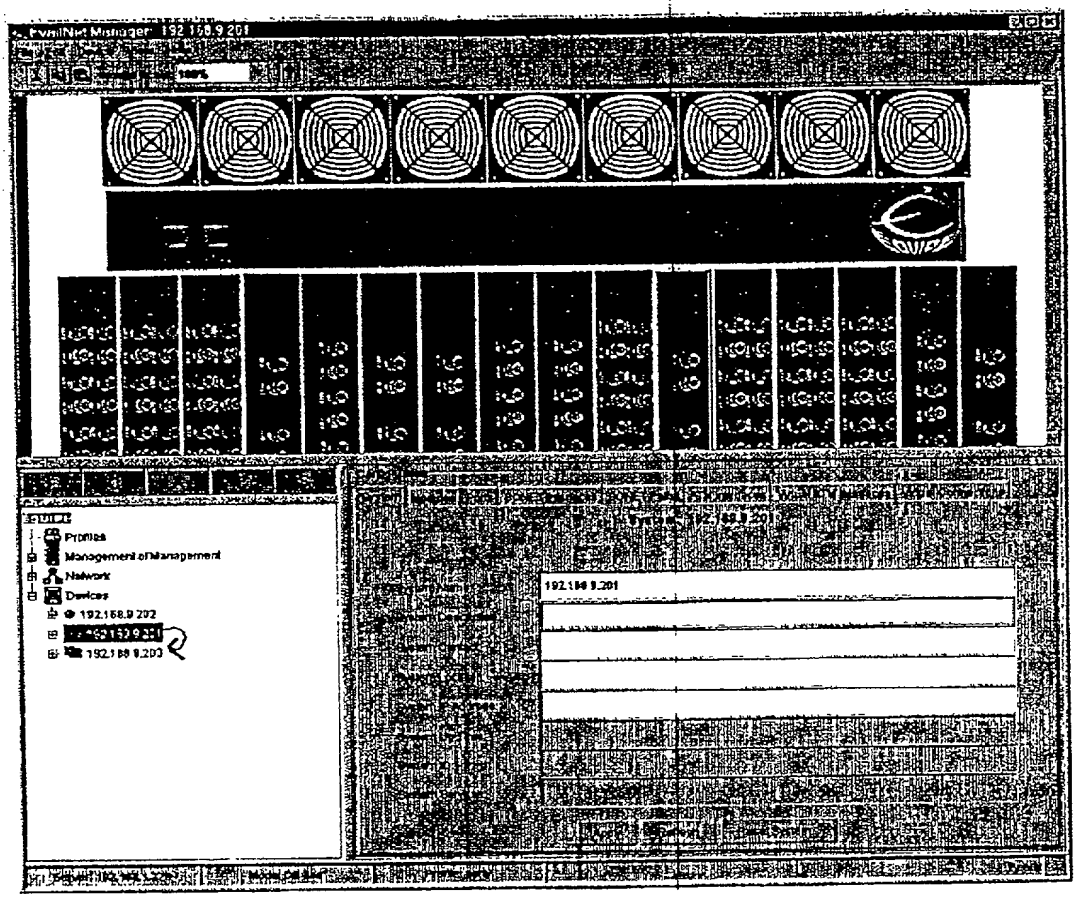

Profiles
Management of Management
Network
Devices
192.168.9.202
192.168.9.203
192.168.9.201

Fig. 6o

895

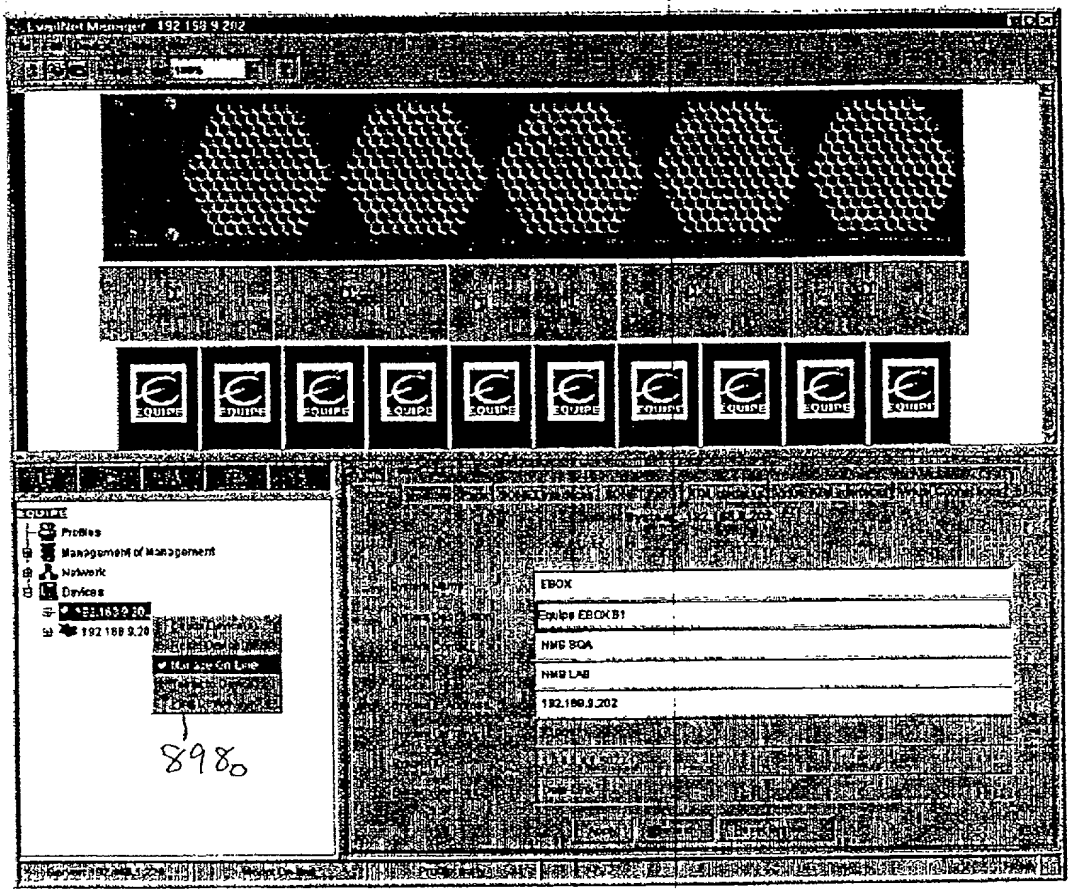

EQUIPE
Profiles
Management of Management
Network
Devices
EBOX
Equipe EBOX B1
NMS SQA
NMS LAB
192.168.9.202
898

Fig. 6p

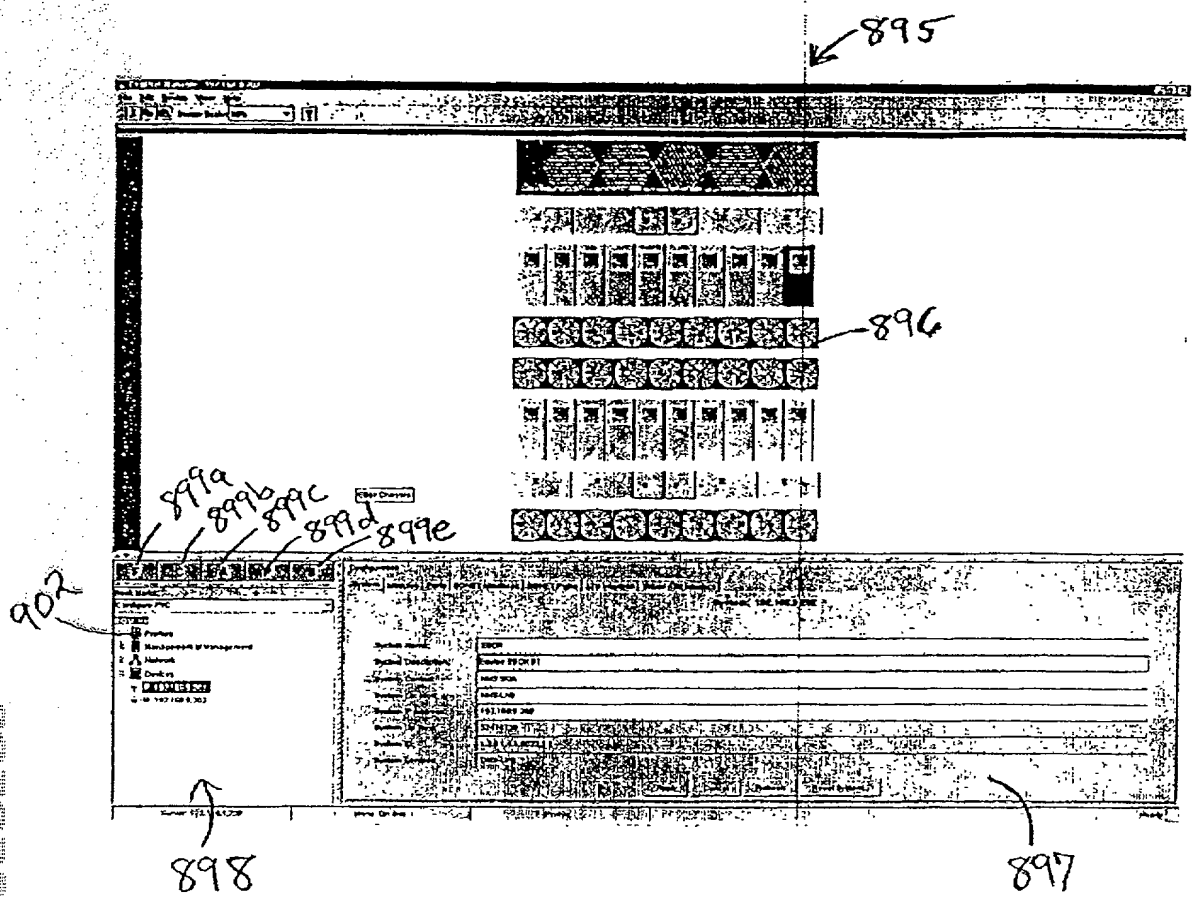


Fig. 7a

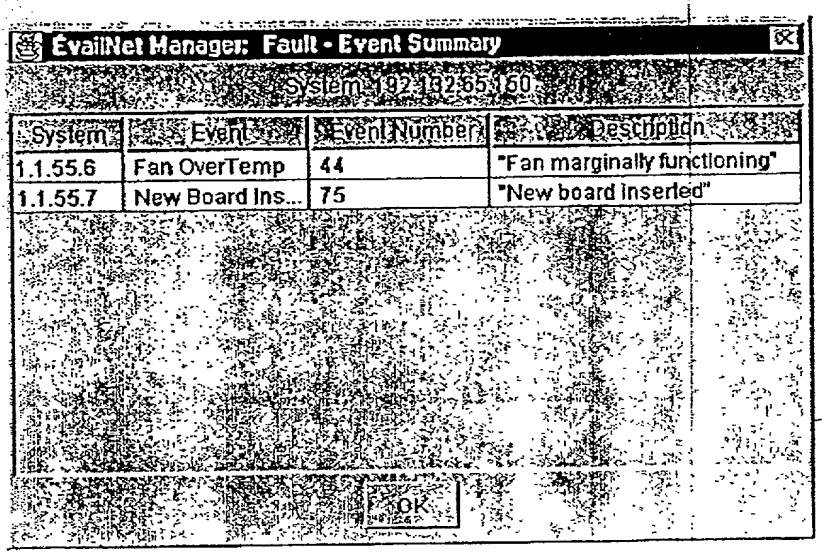


Fig. 7b

DOCKET ALARM

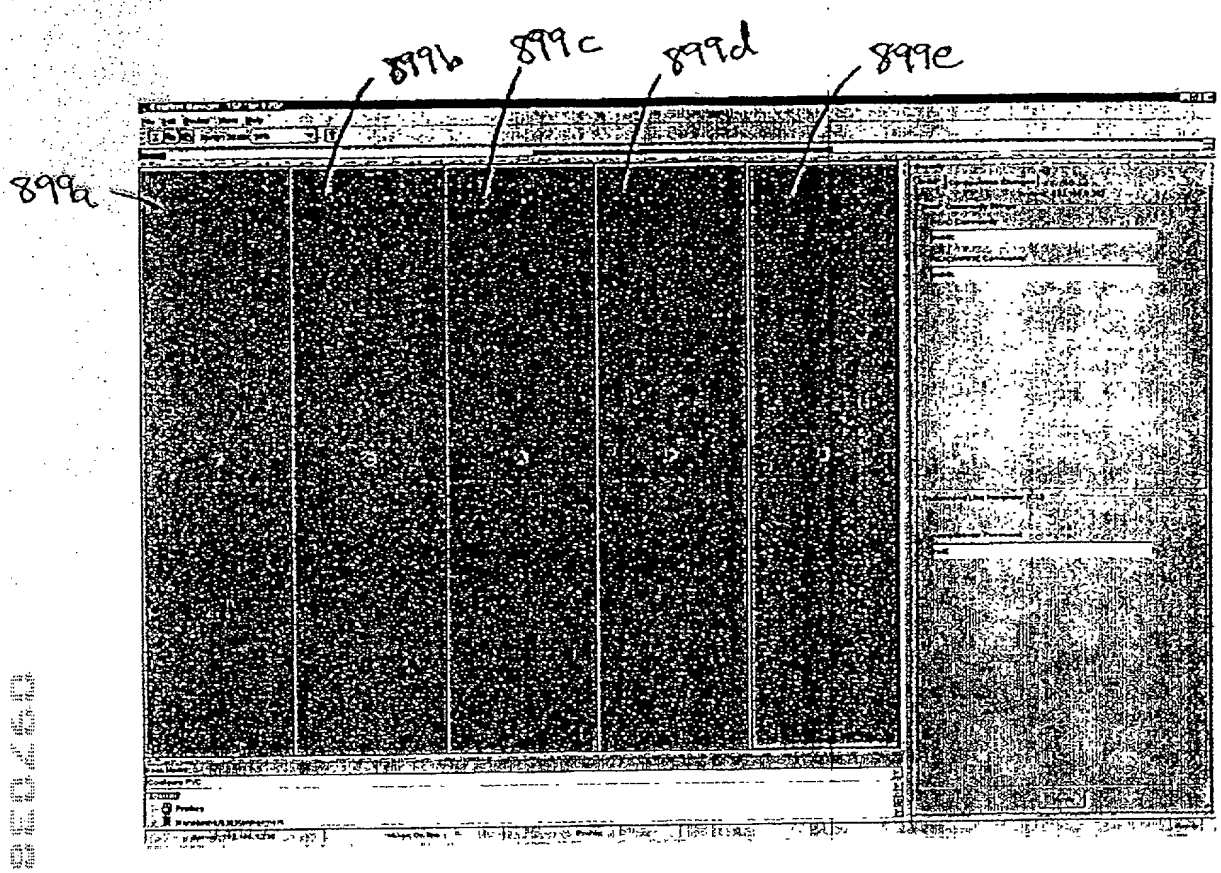


Fig.7c

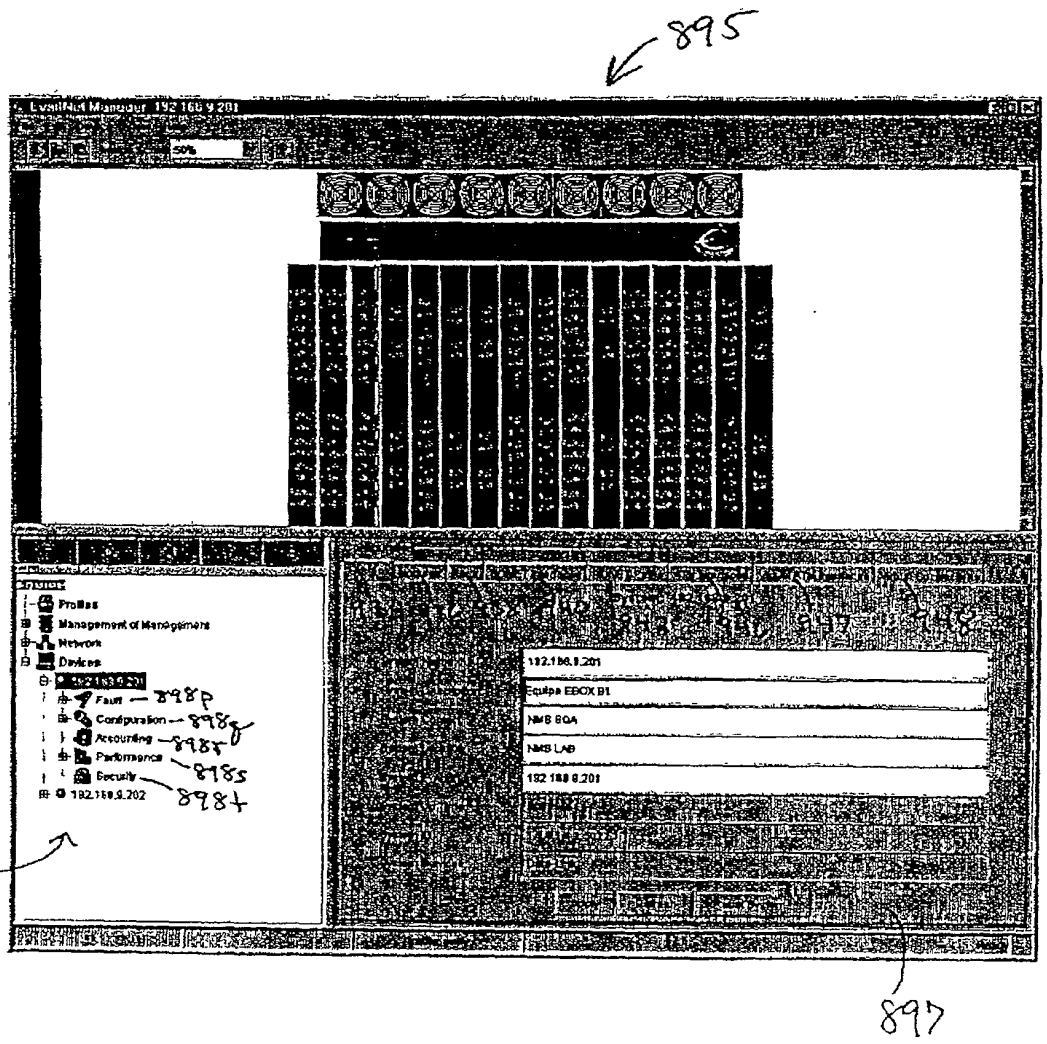

895
Profiles
Management of Managements
Network
Devices
Fault
Configuration
Accounting
Performance
Security
192.168.9.202
898p
898s
898t
898s
898t
898
897

Fig. 7d

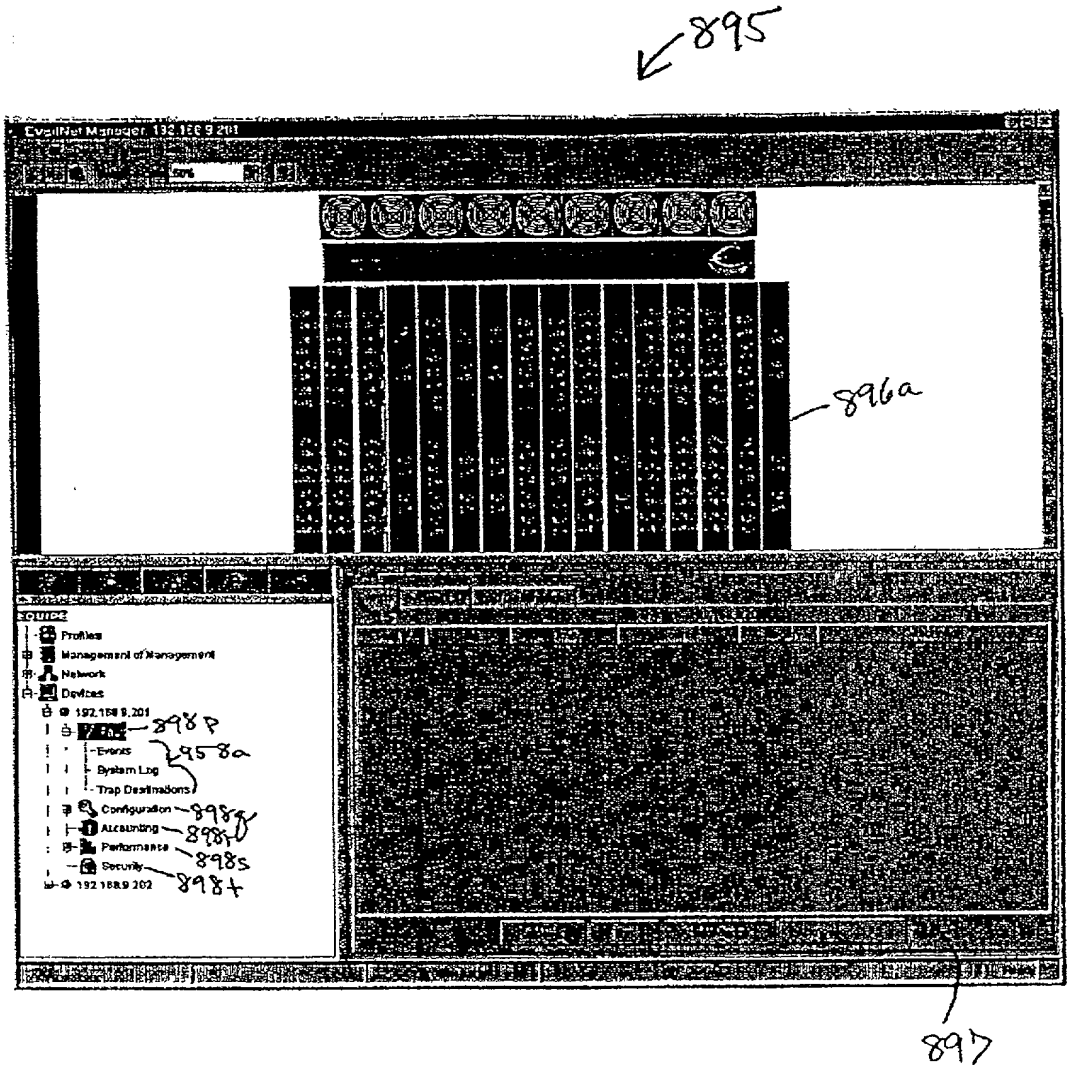

895
896a
Profiles
Management of Management
Network
Devices
898p
Events
System Log
Trap Destinations
958a
Configuration
898g
Accounting
898t
Performance
898s
Security
898t
897

Fig. 7e

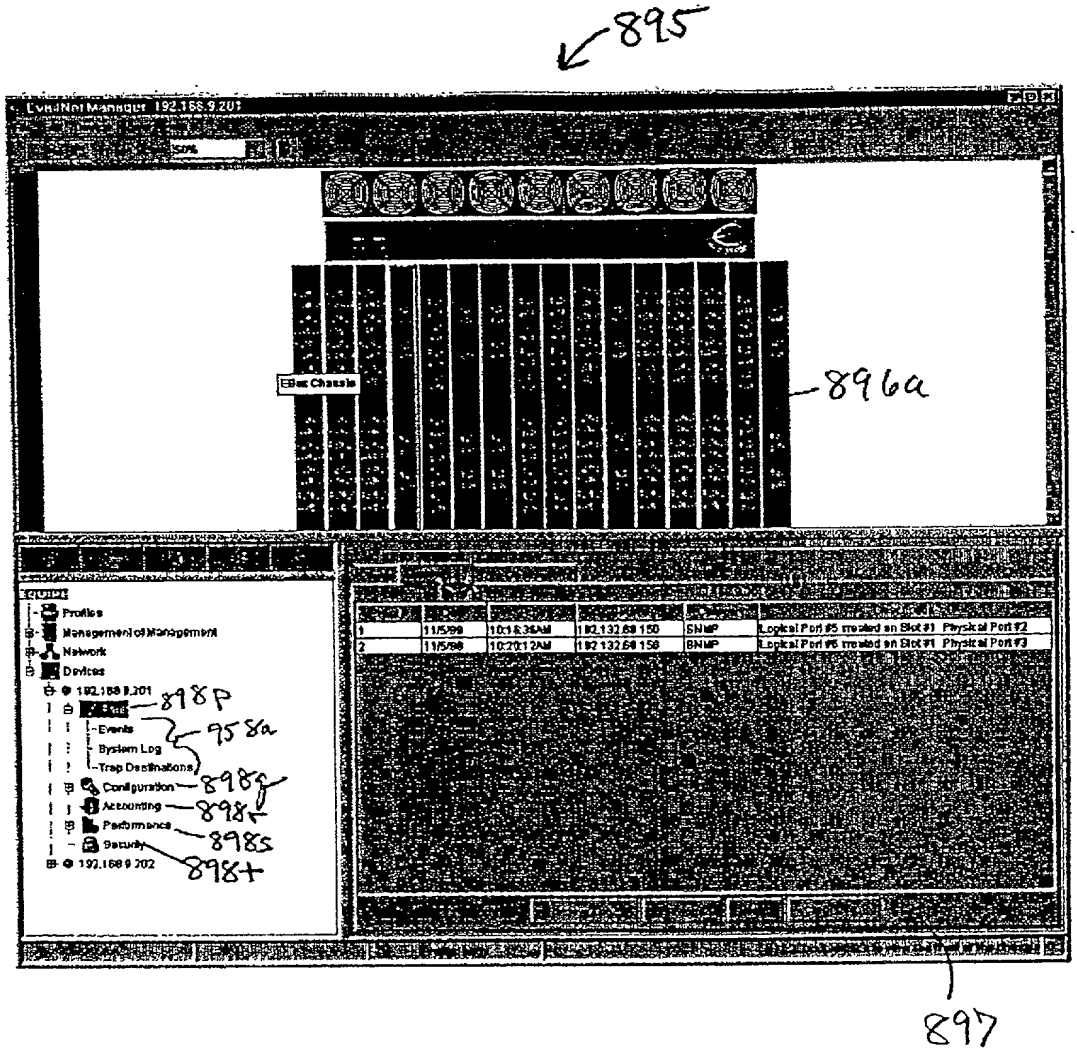

895
896a
898p
958a
898g
898t
898s
898t
897
Profiles
Management Management
Network
Devices
Events
System Log
Trap Destinations
Configuration
Accounting
Performance
Security

Fig. 7f

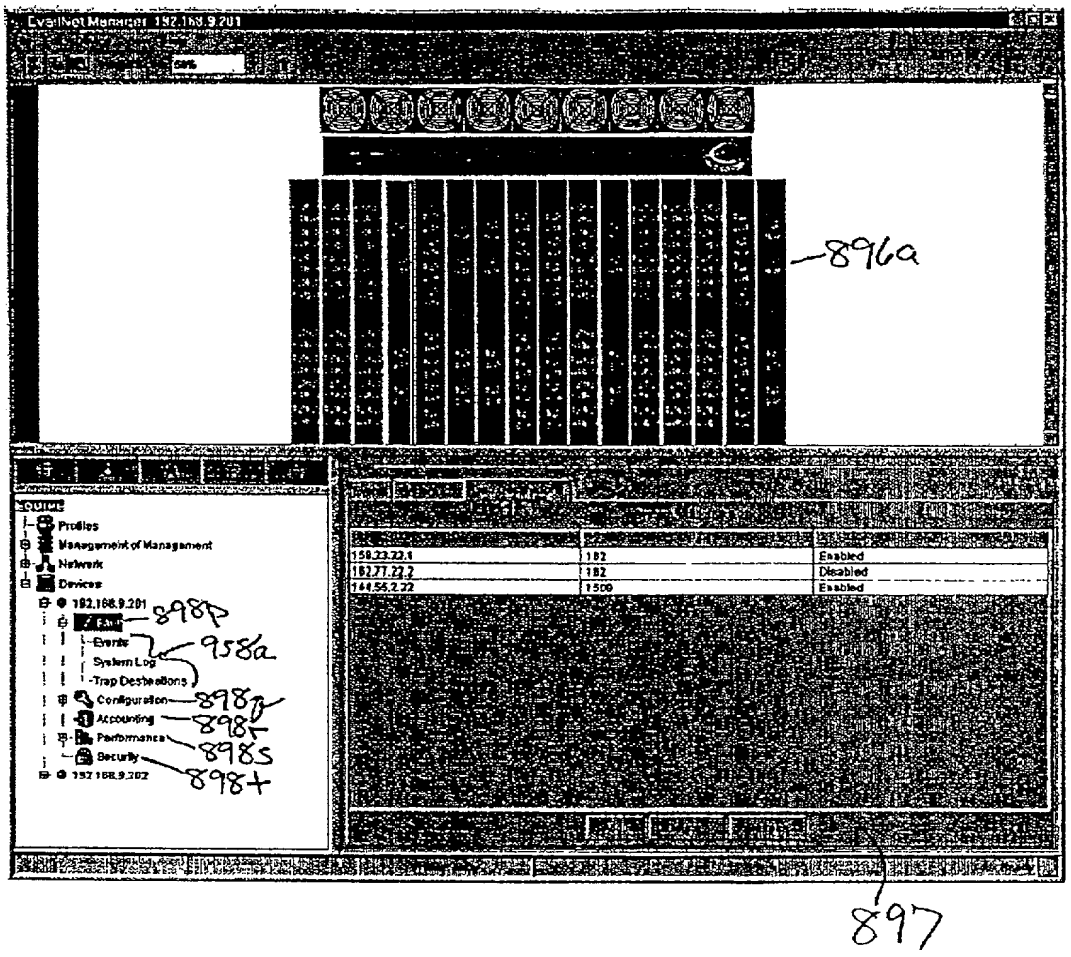

895
896a
Profiles
Management of Management
Network
Devices
182.168.9.201
Fault
Events
System Log
Trap Destinations
Configuration
Accounting
Performance
Security
182.168.9.202
898b
958a
898g
898f
898s
898t
897

Fig. 7g

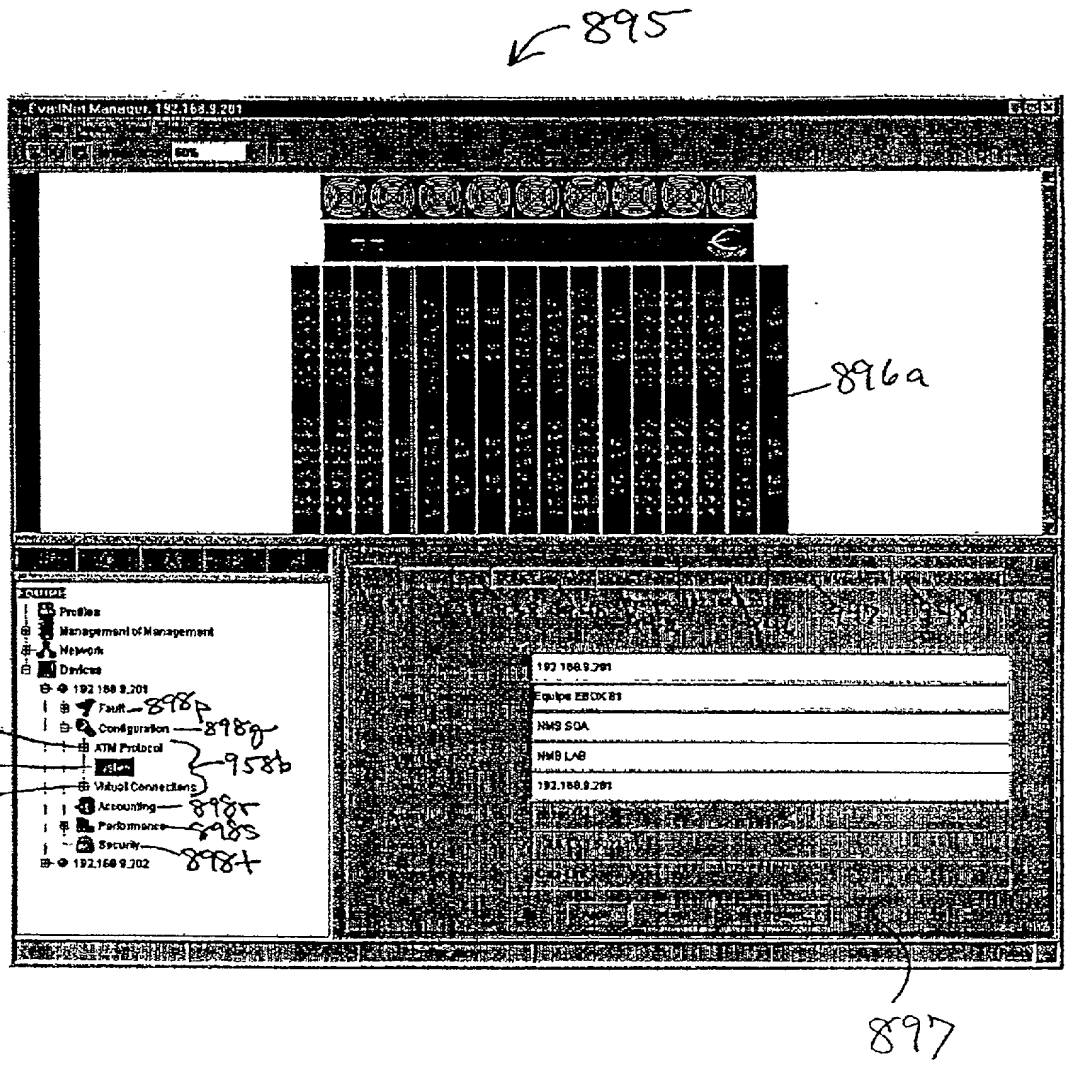

895
896a
958c
958d
958e
958b
898f
898g
898r
898s
898t
897
Profiles
Management of Management
Network
Devices
Fault
Configuration
ATM Protocol
Virtual Connections
Accounting
Performance
Security
NMS SOA
NMS LAB

Fig. 7h

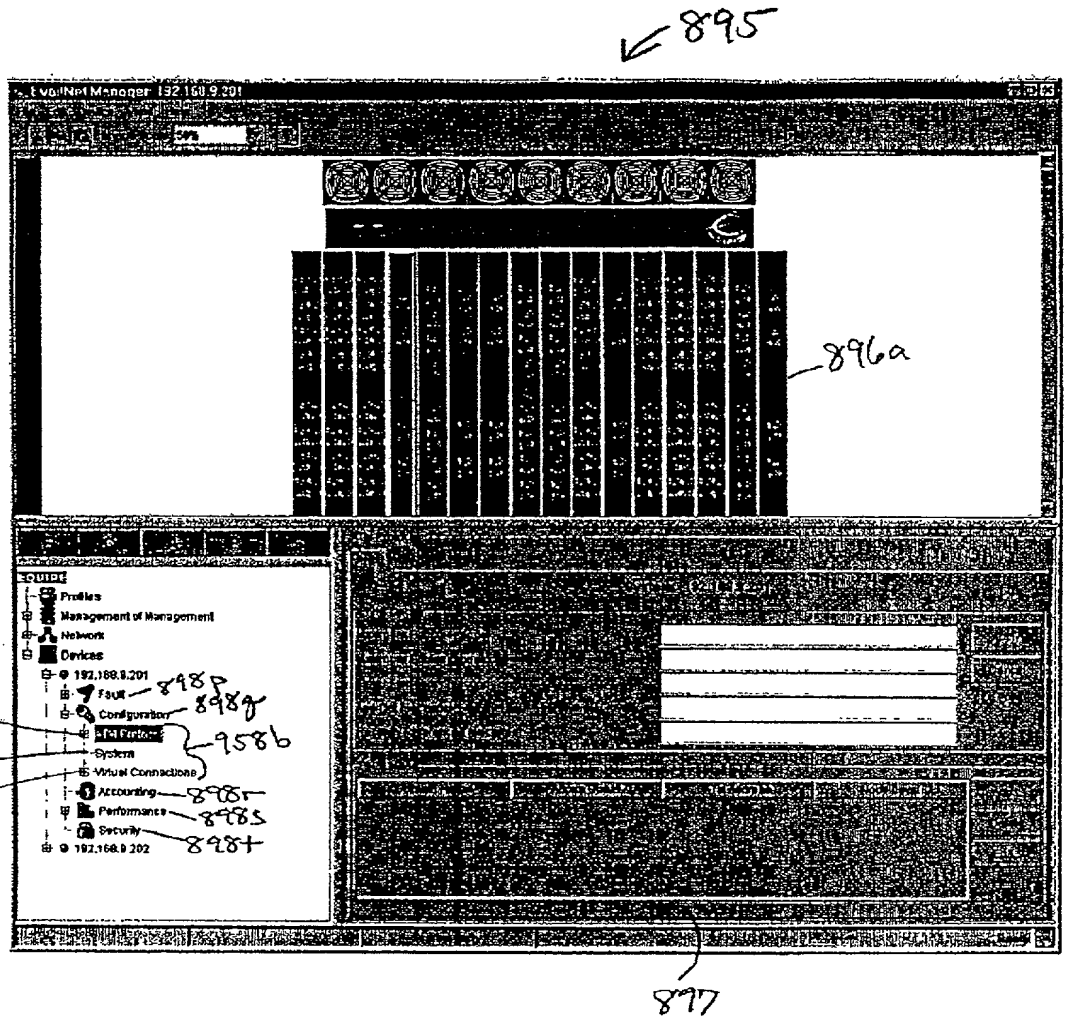

895
896a
958c
958d
958e
958b
898p
898q
898r
898s
898t
897
Profiles
Management of Management
Network
Devices
192.168.9.201
Fault
Configuration
Virtual Connections
Accounting
Performance
Security
192.168.9.202

Fig. 7i

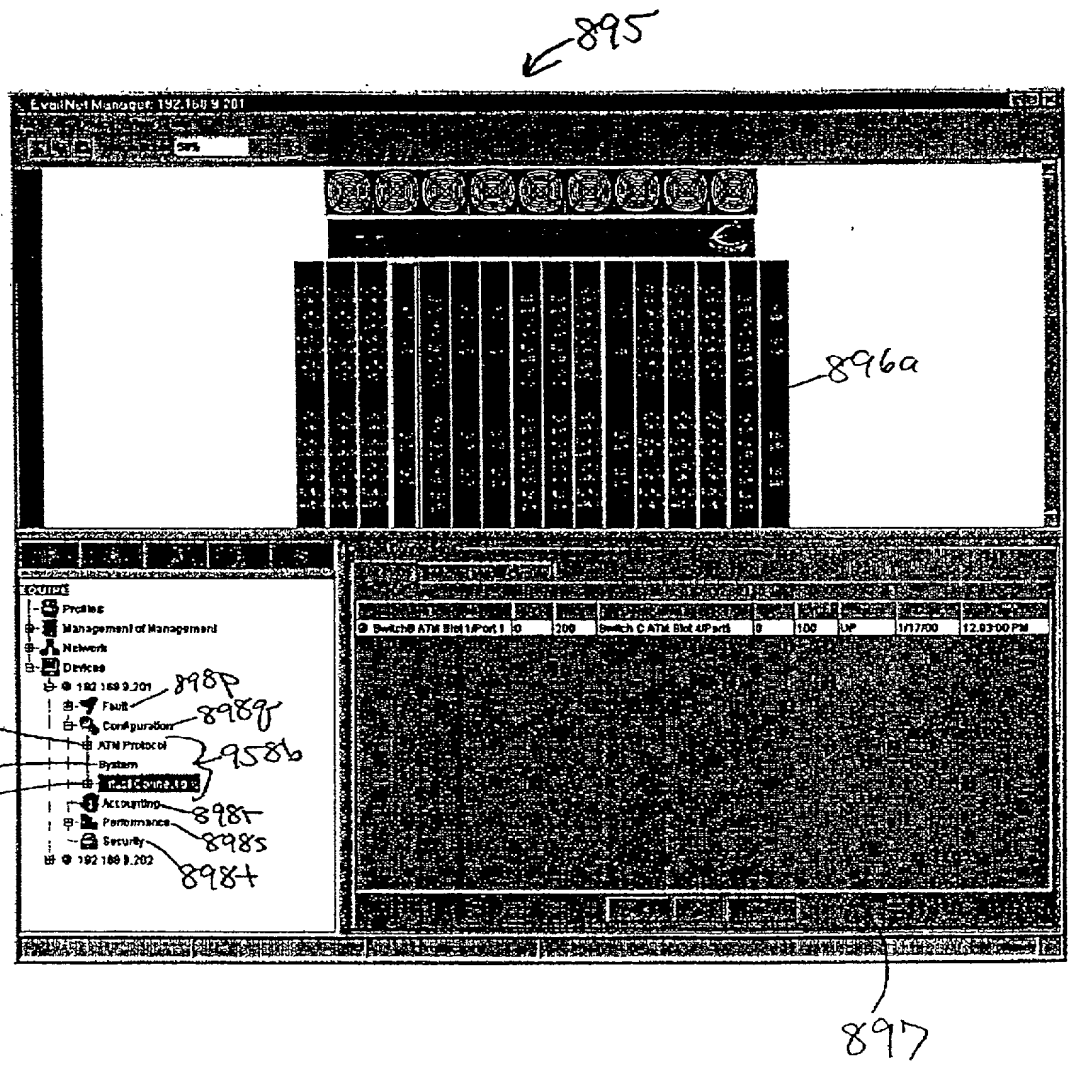


Fig. 7j

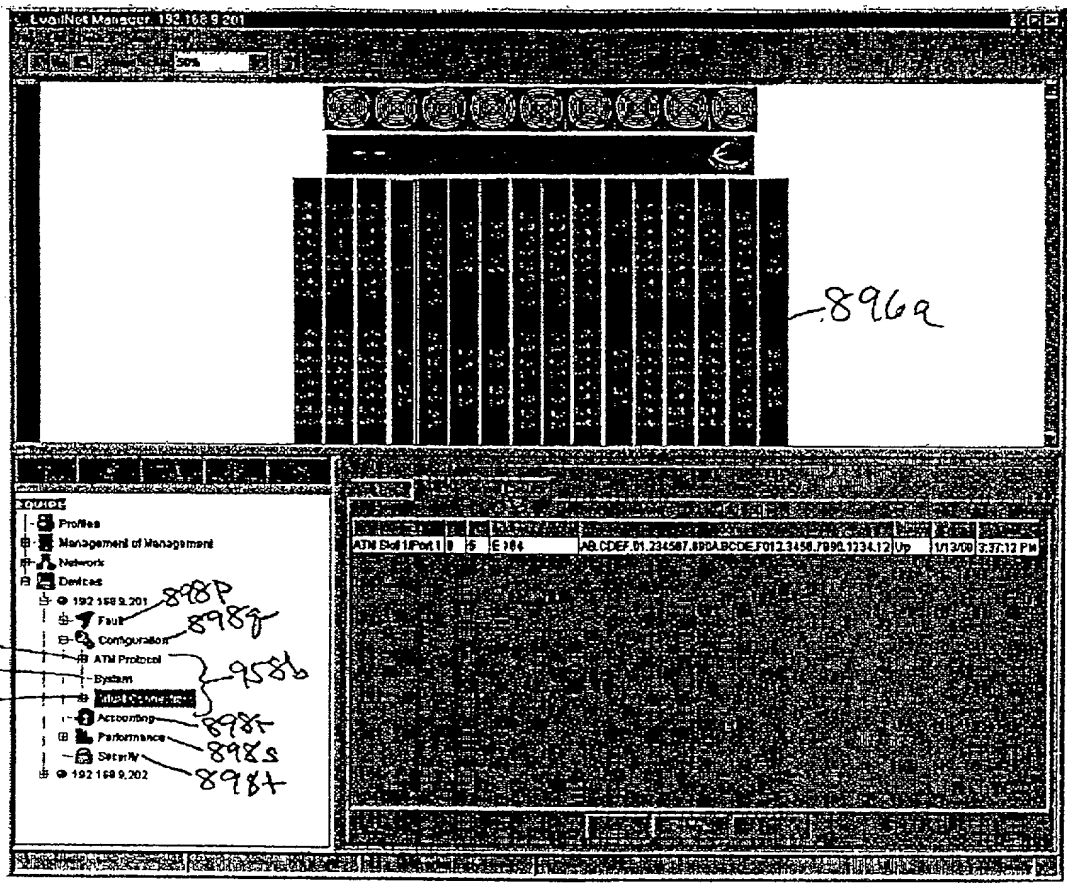

895
LocalNet Manager 192.168.9.201
896a
Profiles
Management of Management
Network
Devices
192.168.9.201
Fault
Configuration
ATM Protocol
System
Accounting
Performance
Security
192.168.9.202
898p
898q
958c
958d
958e
958b
898r
898s
898t

Fig. 7K

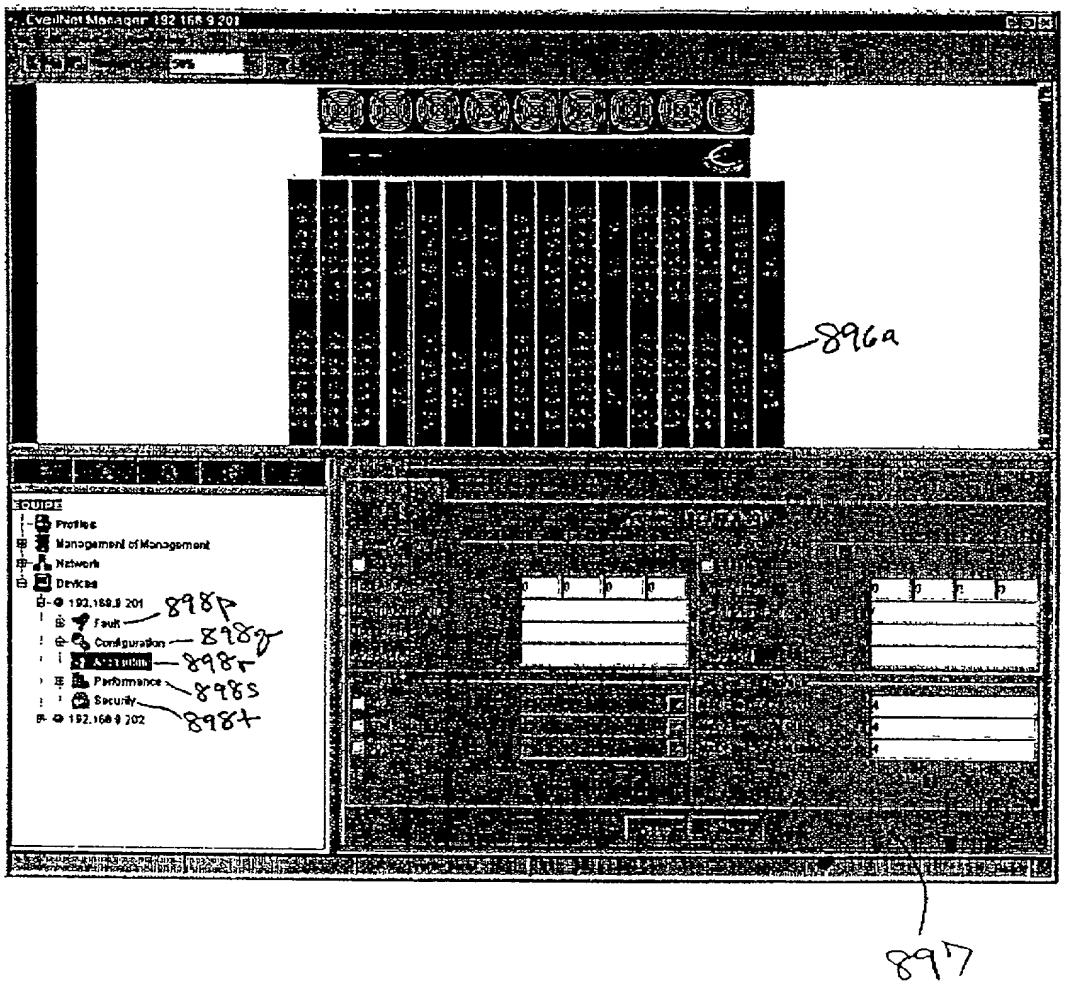

895
896a
897
898p
898q
898r
898s
898t
Profiles
Management of Management
Network
Devices
Fault
Configuration
Accounting
Performance
Security

Fig. 7L

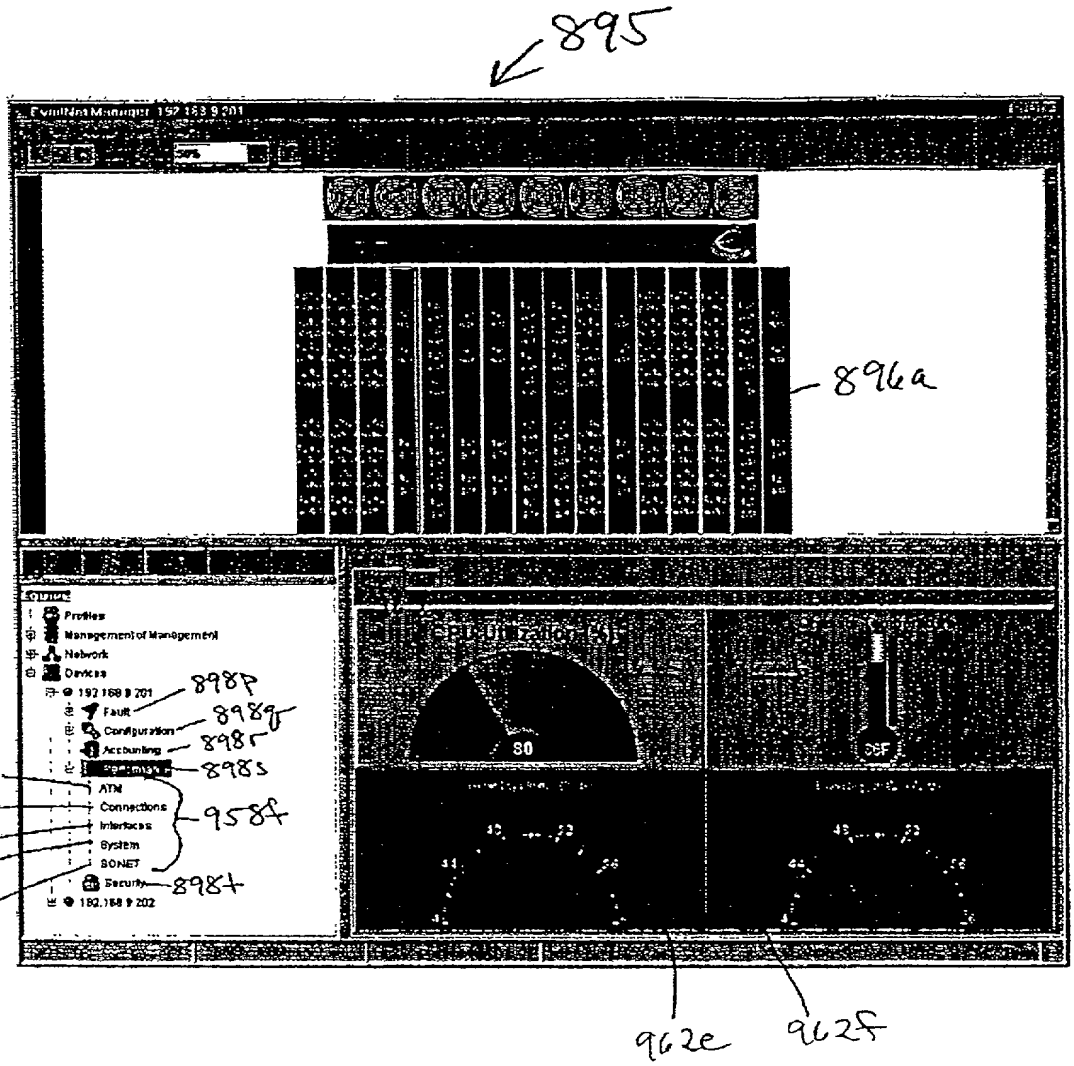

895
896a
Profiles
Management of Management
Network
Devices
192.168.9.201
Fault
Configuration
Accounting
Connections
Interfaces
System
SONET
Security
192.168.9.202
8987
8989
8985
8983
9584
9589
9588
9587
9586
9585
9584
80
962e
962f

Fig. 7m

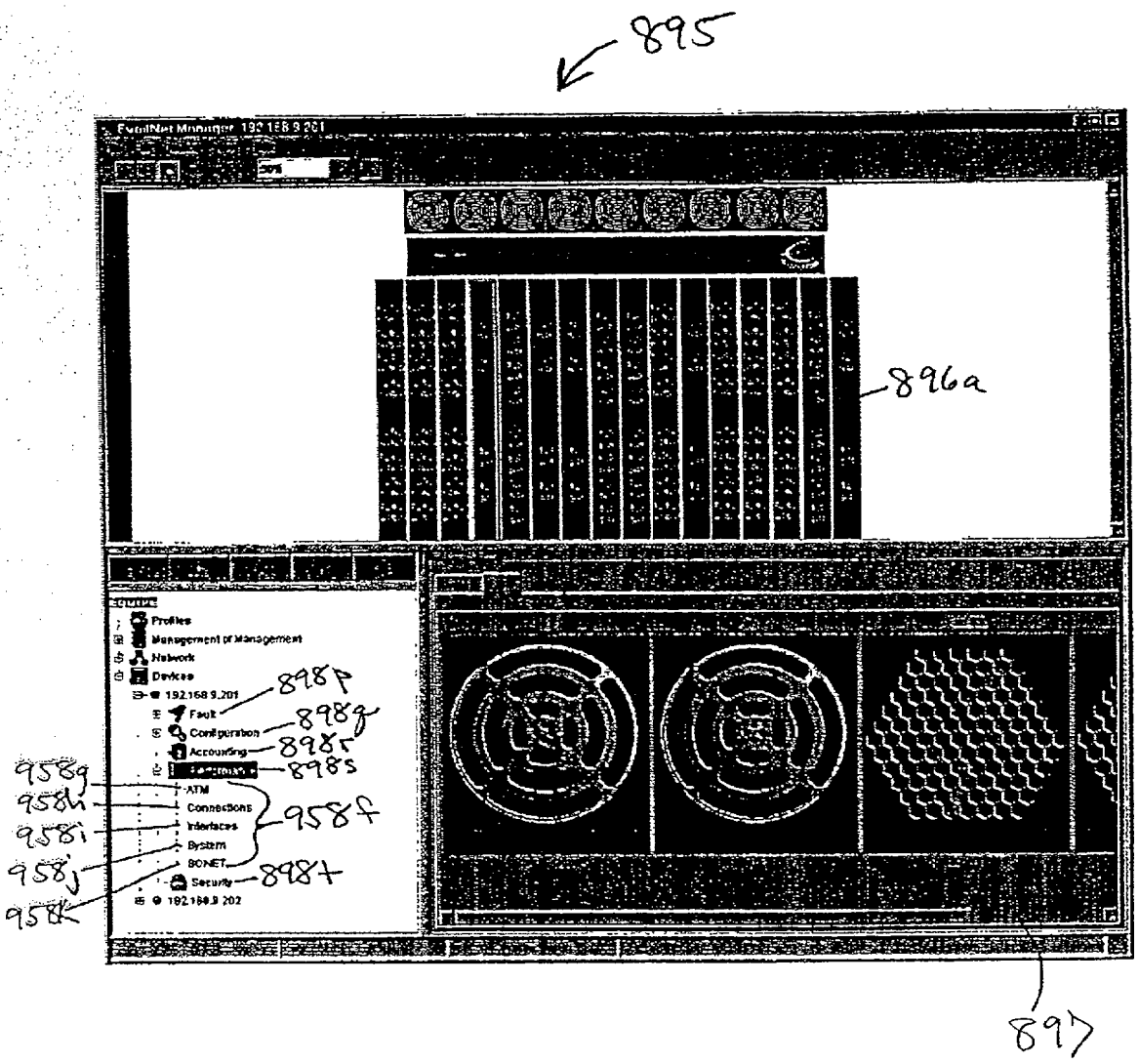

895
896a
Profiles
Management of Management
Network
Devices
192.168.9.201
Fault
Configuration
Accounting
Connections
Interfaces
System
SONET
Security
192.168.9.202
898f
898g
898r
898s
898t
958g
958h
958i
958j
958k
958f
897

Fig. 7n

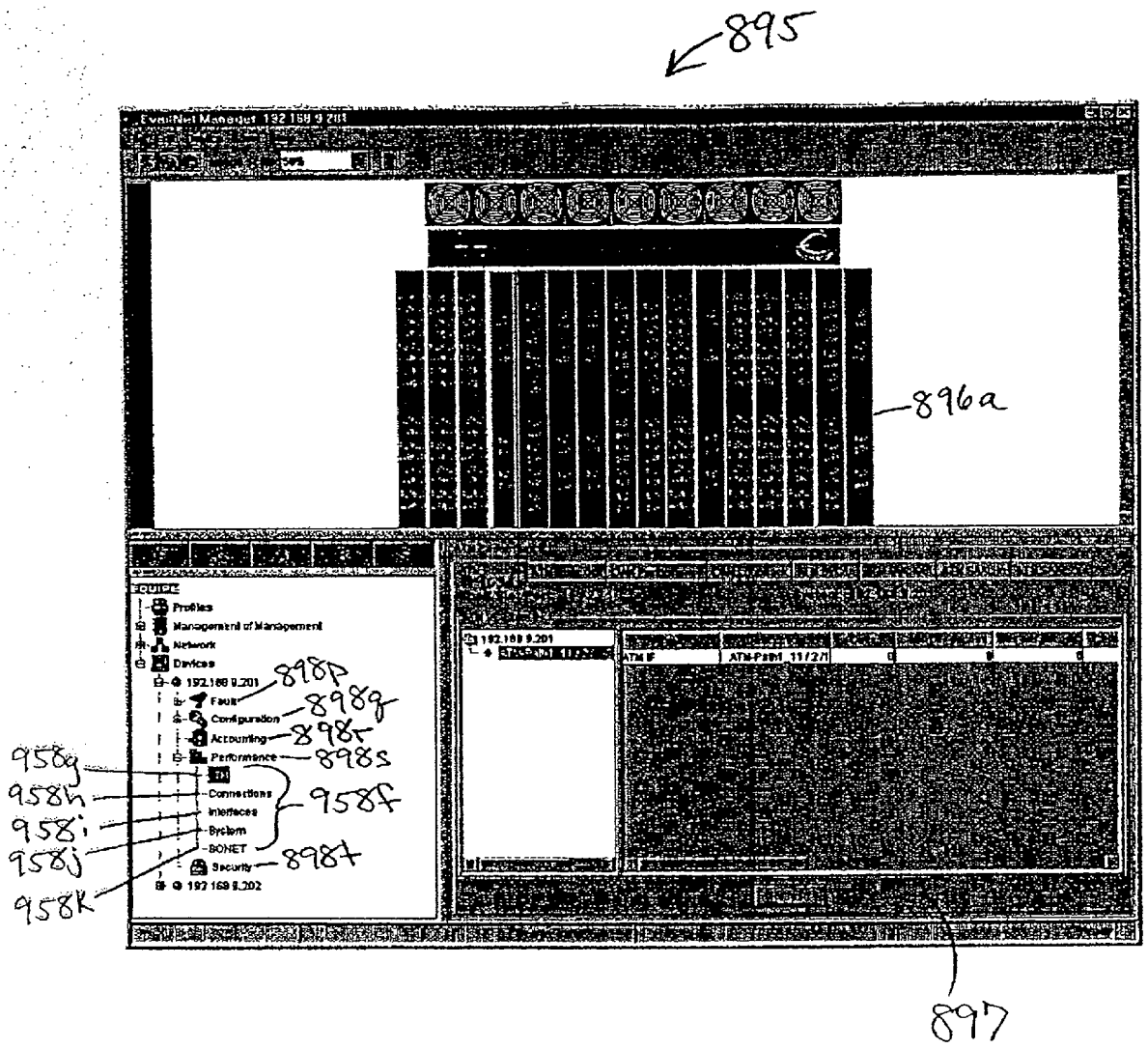

895
896a
898p
898q
898r
898s
958g
958h
958i
958j
958k
958f
8987
897

Fig. 70

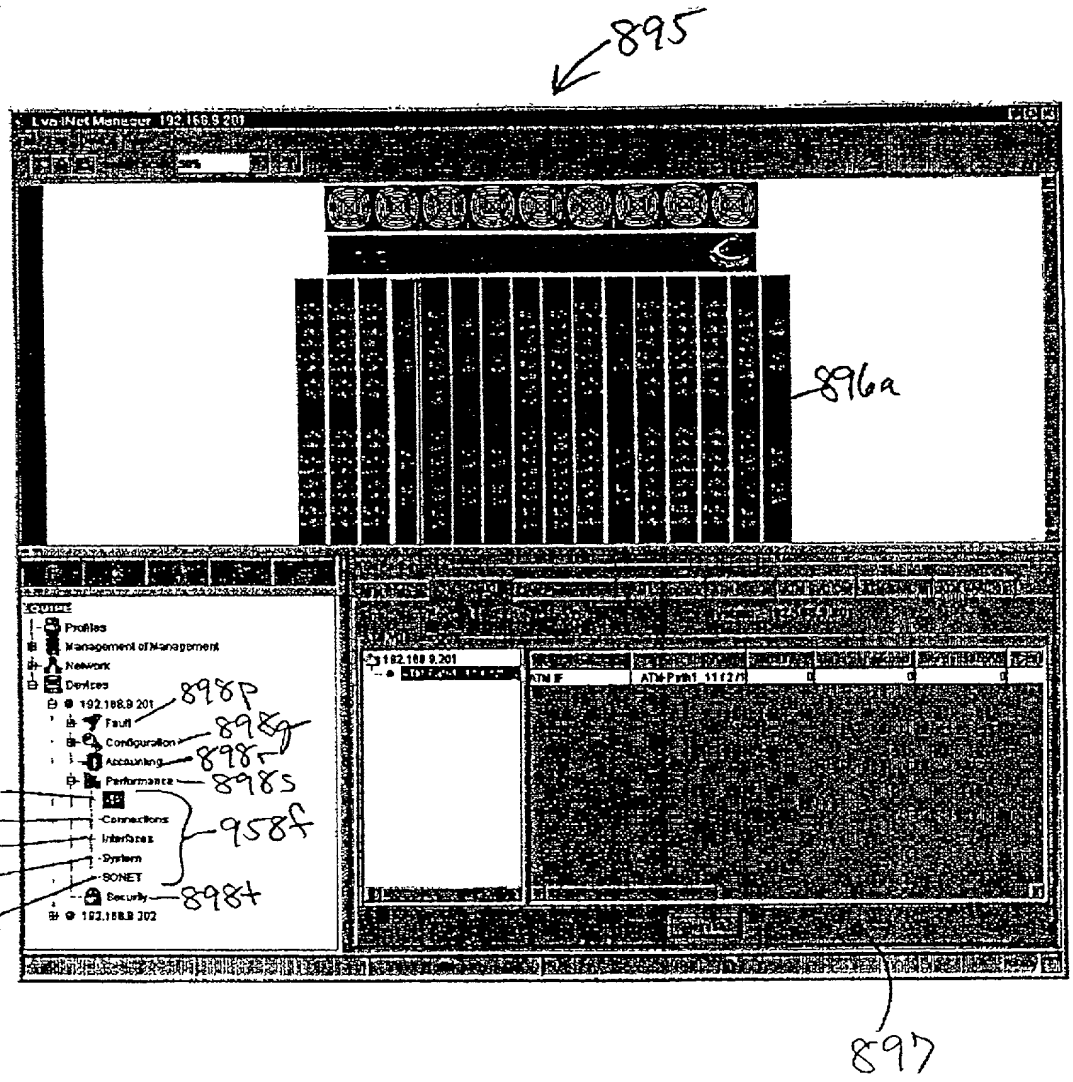

895
896a
898p
898q
898r
898s
958g
958h
958i
958j
958K
958f
898t
897

Fig. 7p

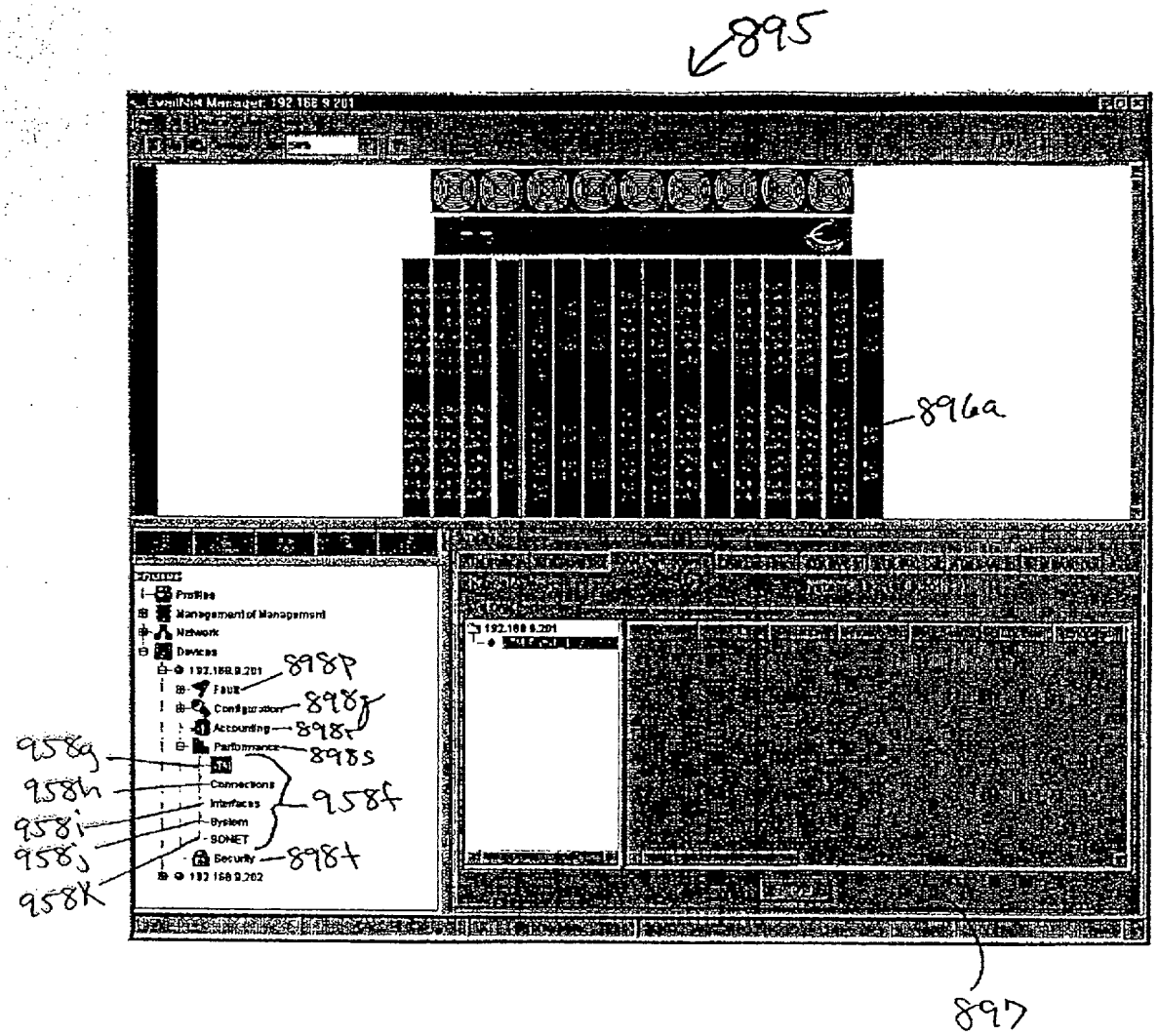

895
896a
898p
898r
898s
958g
958h
958i
958j
958k
958f
898t
897

Fig. 7g

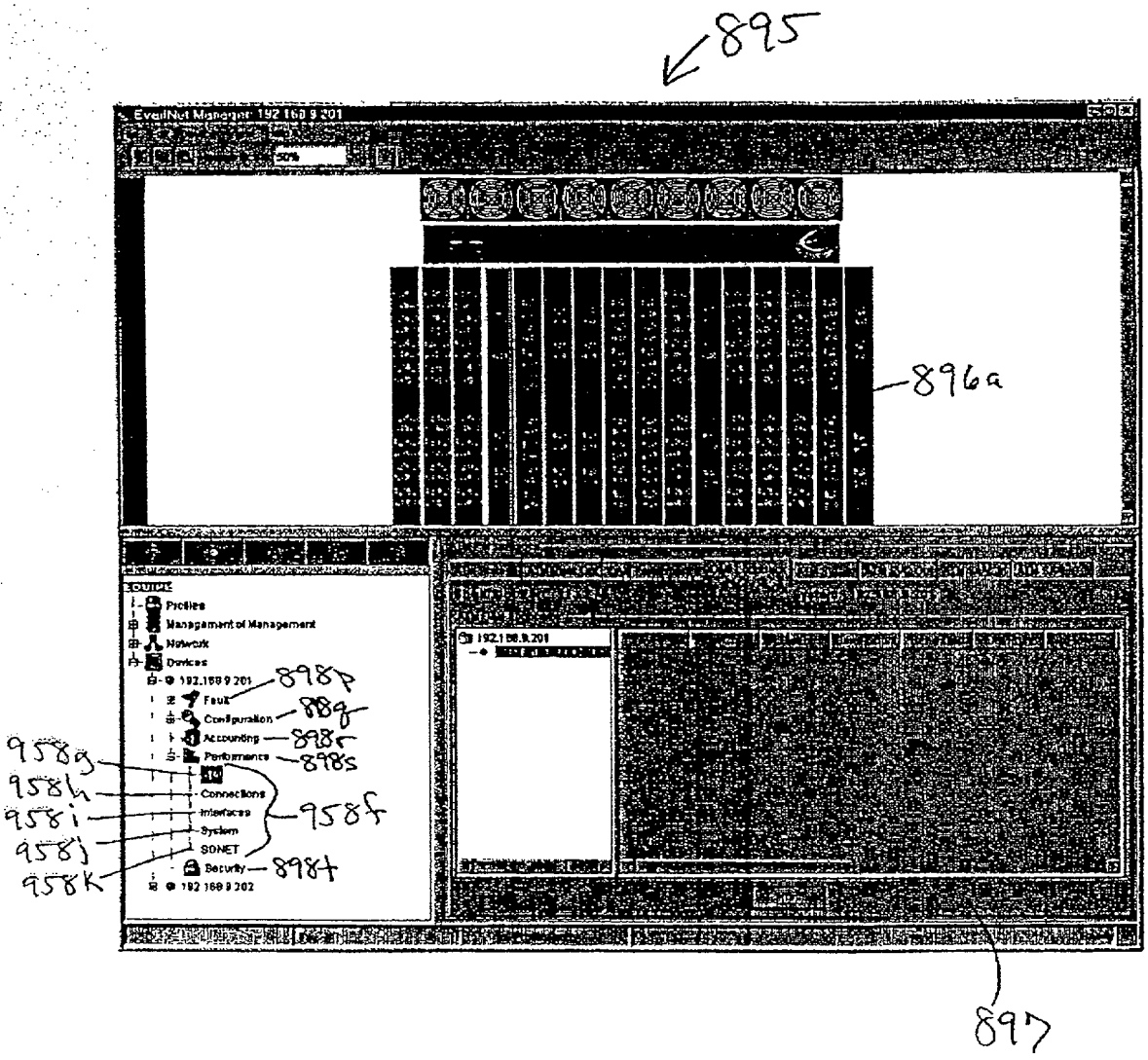

895
896a
897
898p
898q
898r
898s
898t
958f
958g
958h
958i
958j
958k
Profiles
Management of Management
Network
Devices
Fault
Configuration
Accounting
Performance
Connections
Interfaces
System
SONET
Security

Fig. 7F

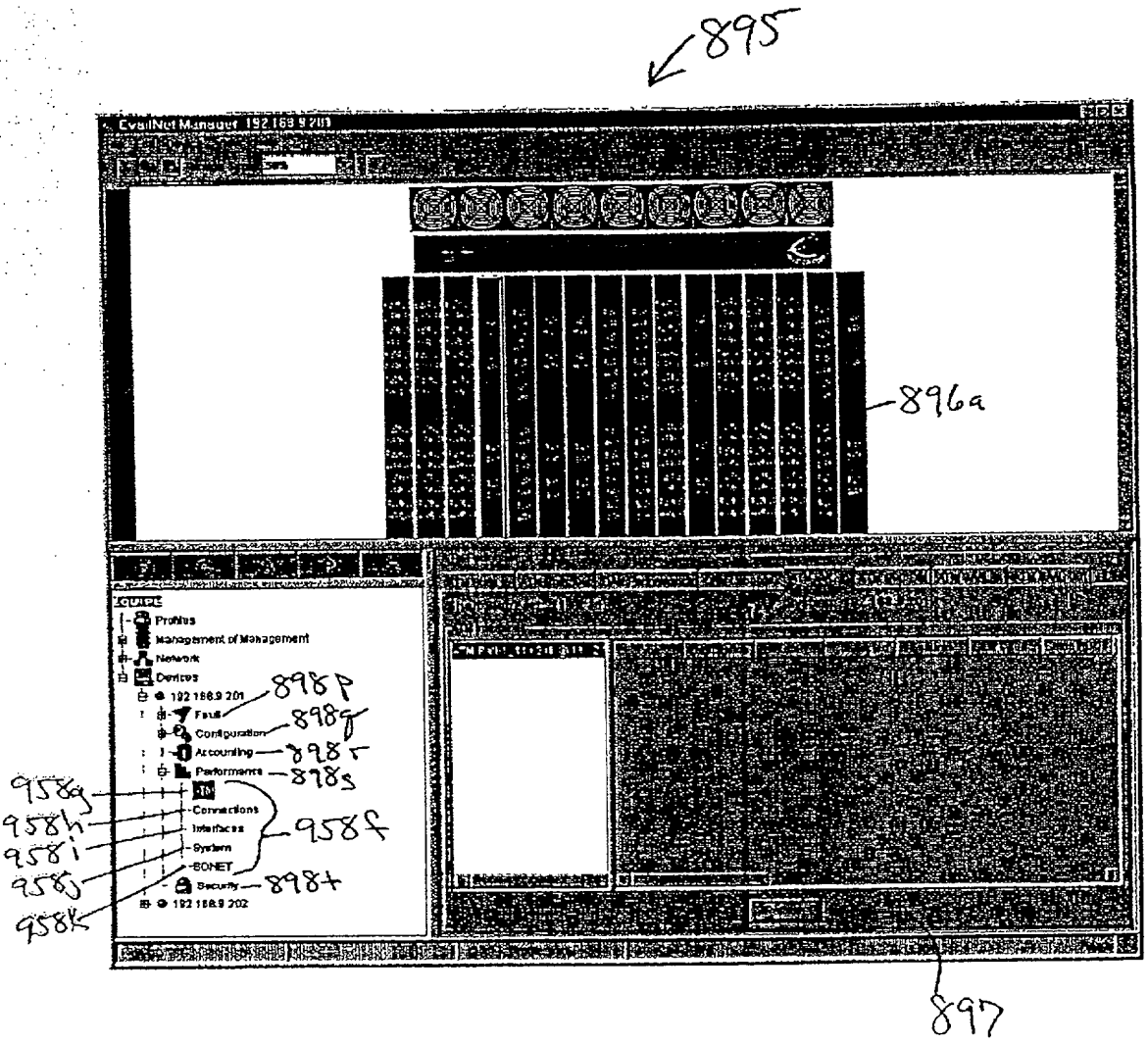

895
896a
897
898p
898q
898r
898s
898t
958e
958f
958g
958h
958i
958j
958k
EQUIPE
Profiles
Management of Management
Network
Devices
Fault
Configuration
Accounting
Performance
SM
Connections
Interfaces
System
SONET
Security

Fig. 7s

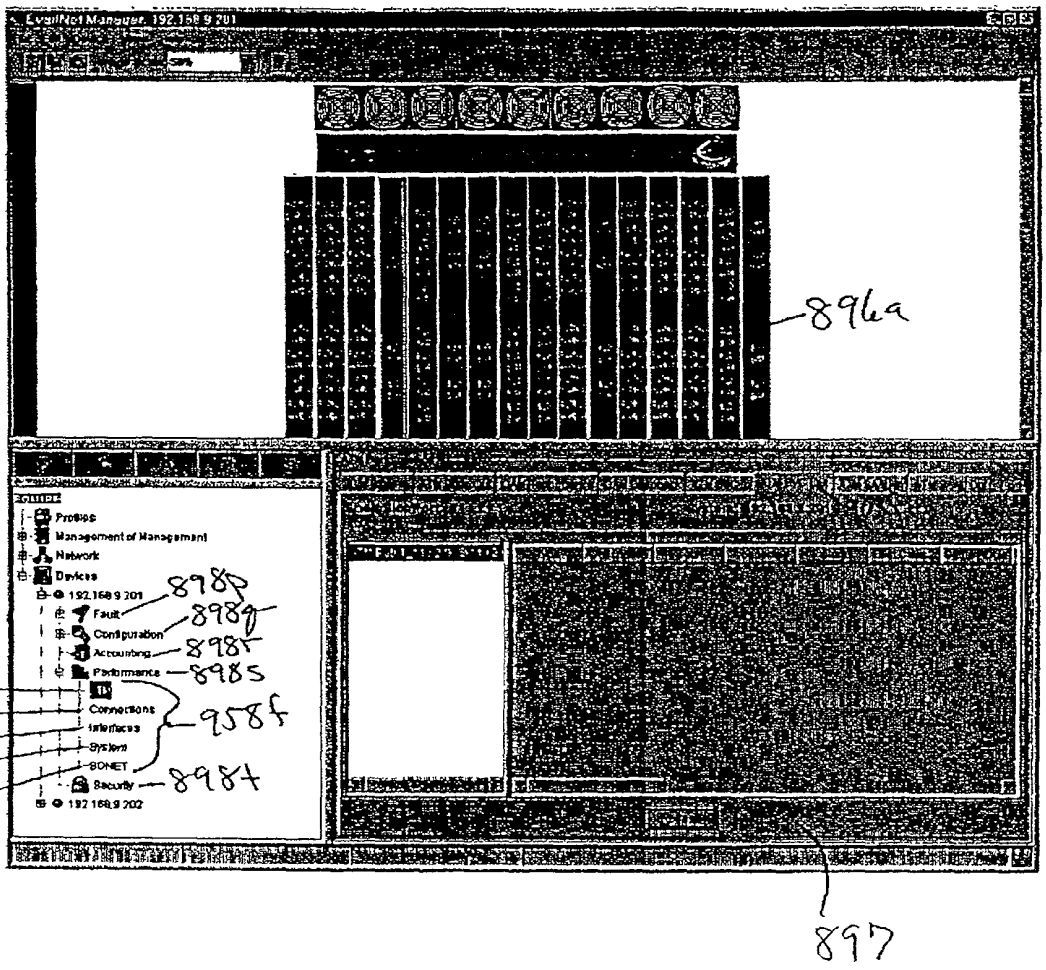

895
896a
Probes
Management of Management
Network
Devices
192.168.9.201
Fault
Configuration
Accounting
Performance
Connections
Interfaces
System
SONET
Security
192.168.9.202
898p
898q
898r
898s
958g
958h
958i
958j
958k
958f
898t
897

Fig. 7t

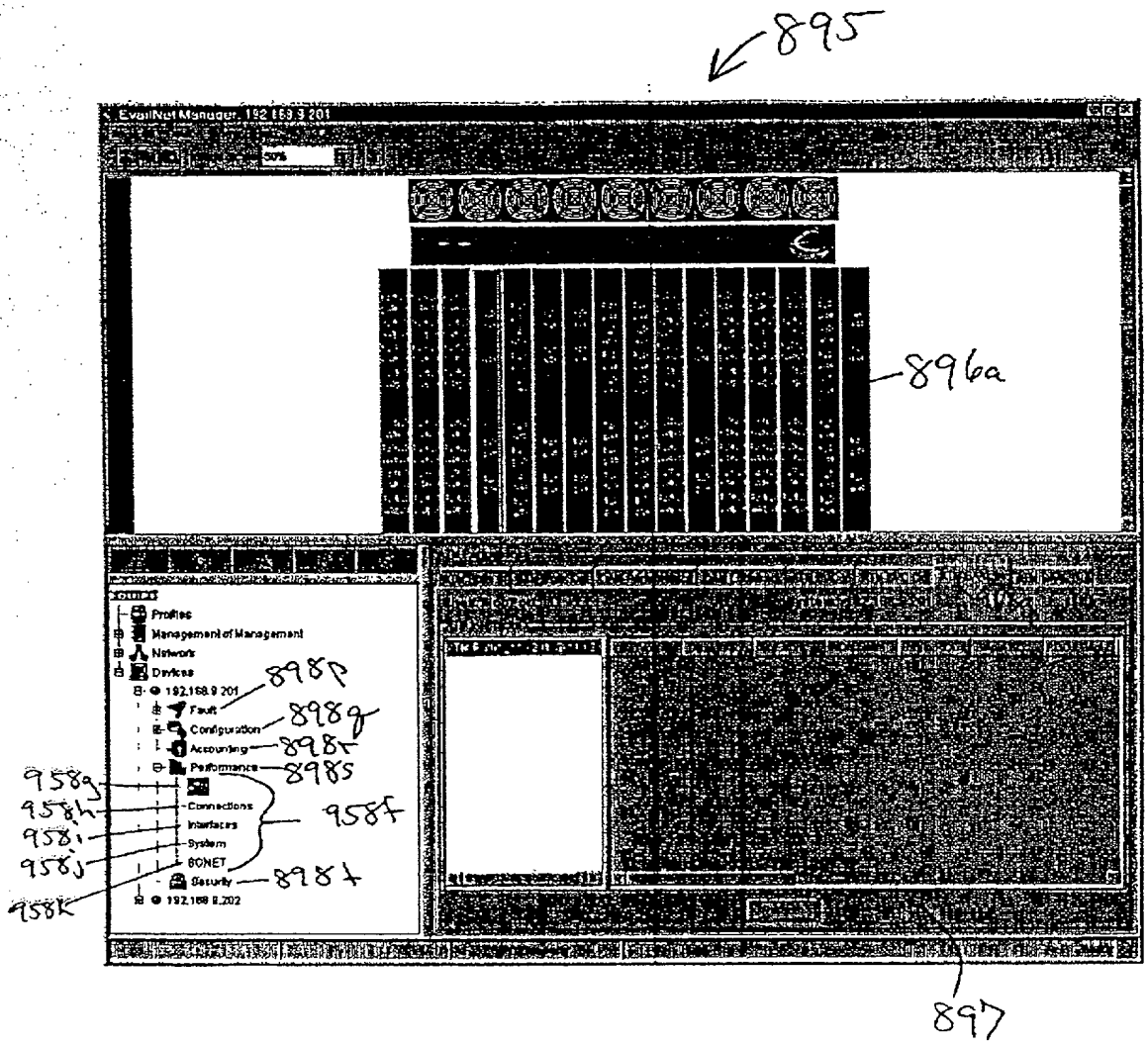

895
896a
Profiles
Management of Management
Network
Devices
Fault
Configuration
Accounting
Performance
Connections
Interfaces
System
SONET
Security
898p
898q
898r
898s
898t
958g
958h
958i
958j
958k
958f
897

Fig. 7u

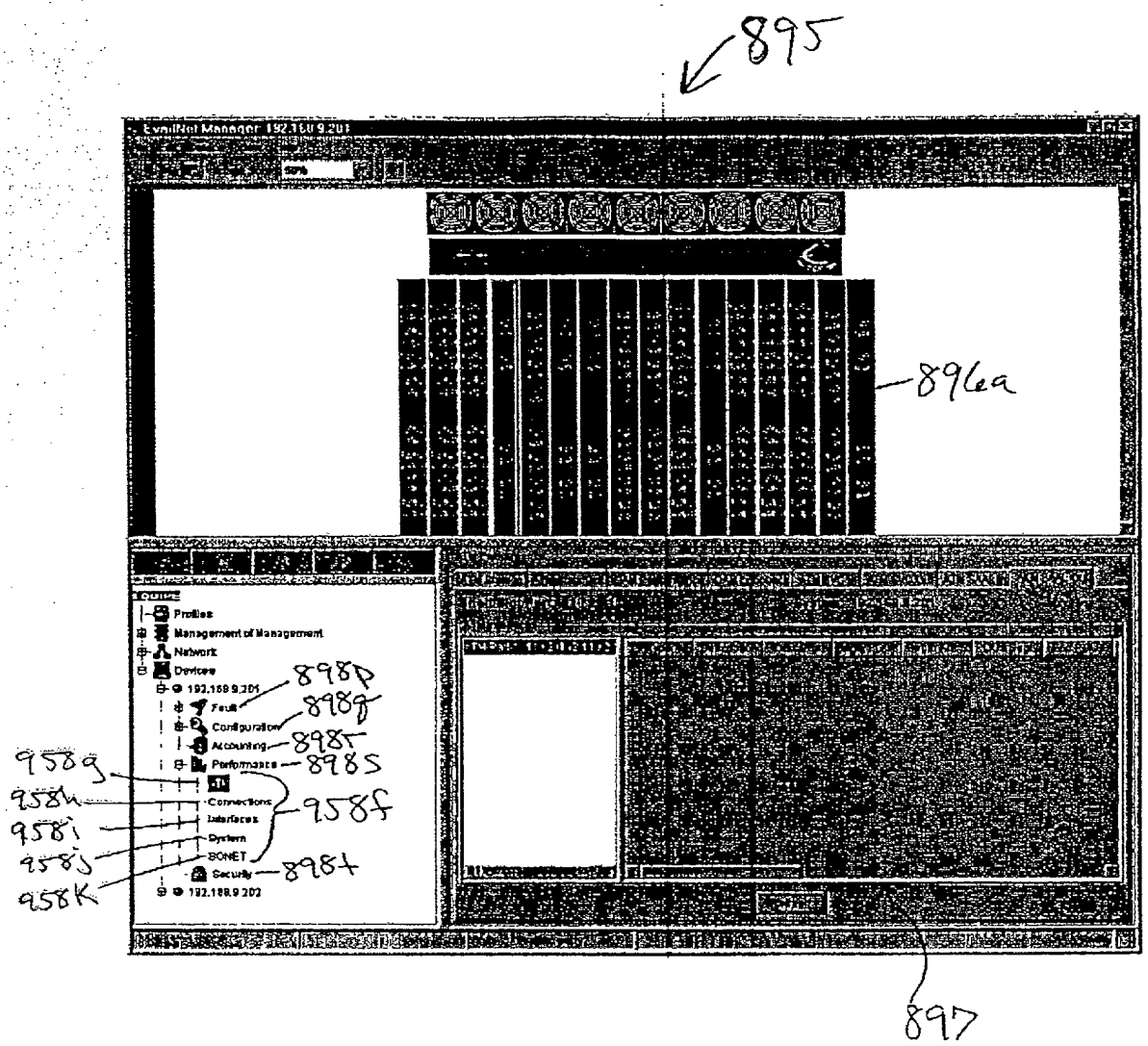

895
896a
Profiles
Management of Management
Network
Devices
Fault
Configuration
Accounting
Performance
Connections
Interfaces
System
SONET
Security
898p
898q
898r
898s
898t
958g
958h
958i
958j
958k
958f
897

Fig. 7v

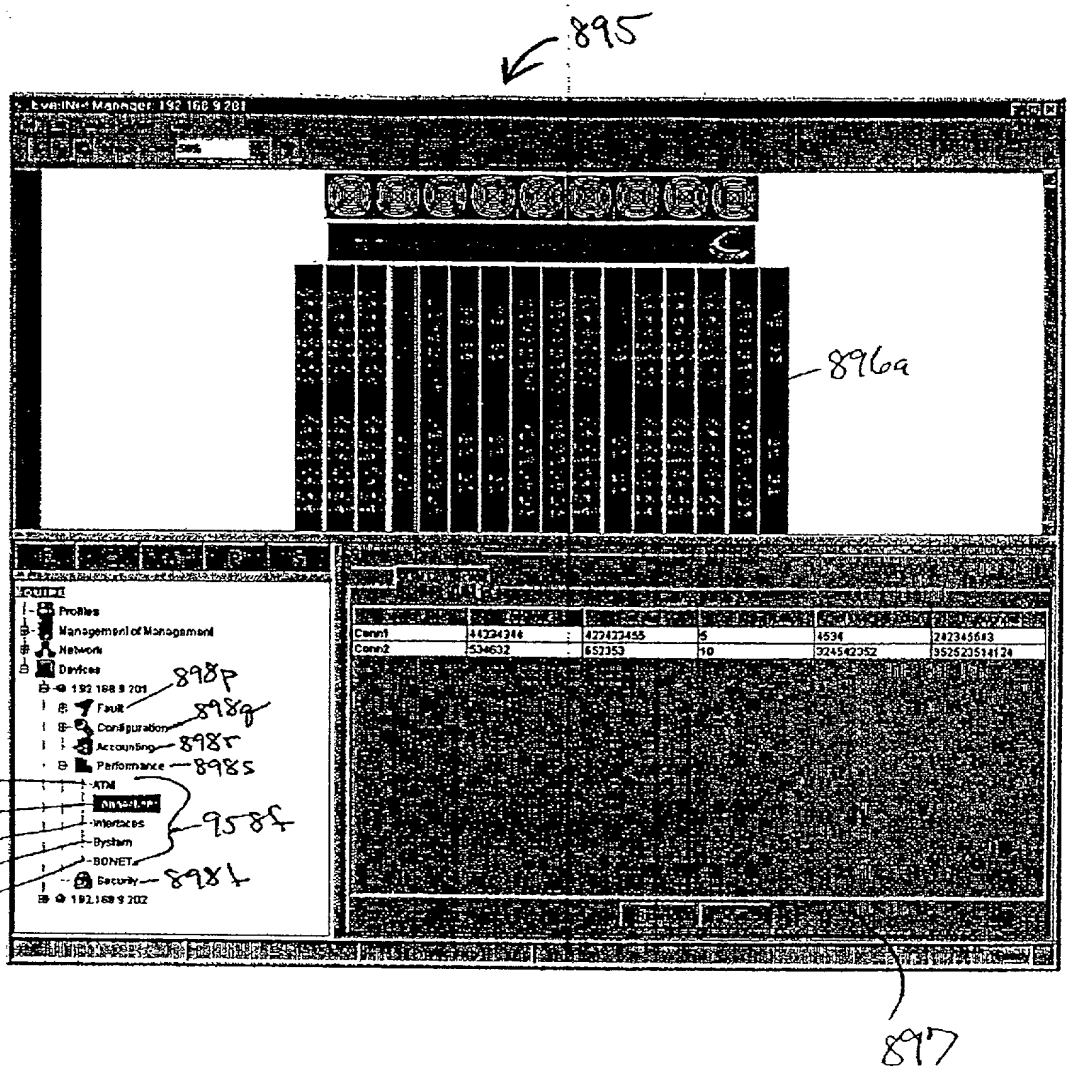

895
896a
Profiles
Management of Management
Network
Devices
Fault
Configuration
Accounting
Performance
ATM
Interfaces
System
SONET
Security
898p
898q
898r
898s
898t
958g
958h
958i
958j
958k
958f
897

Fig. 7w

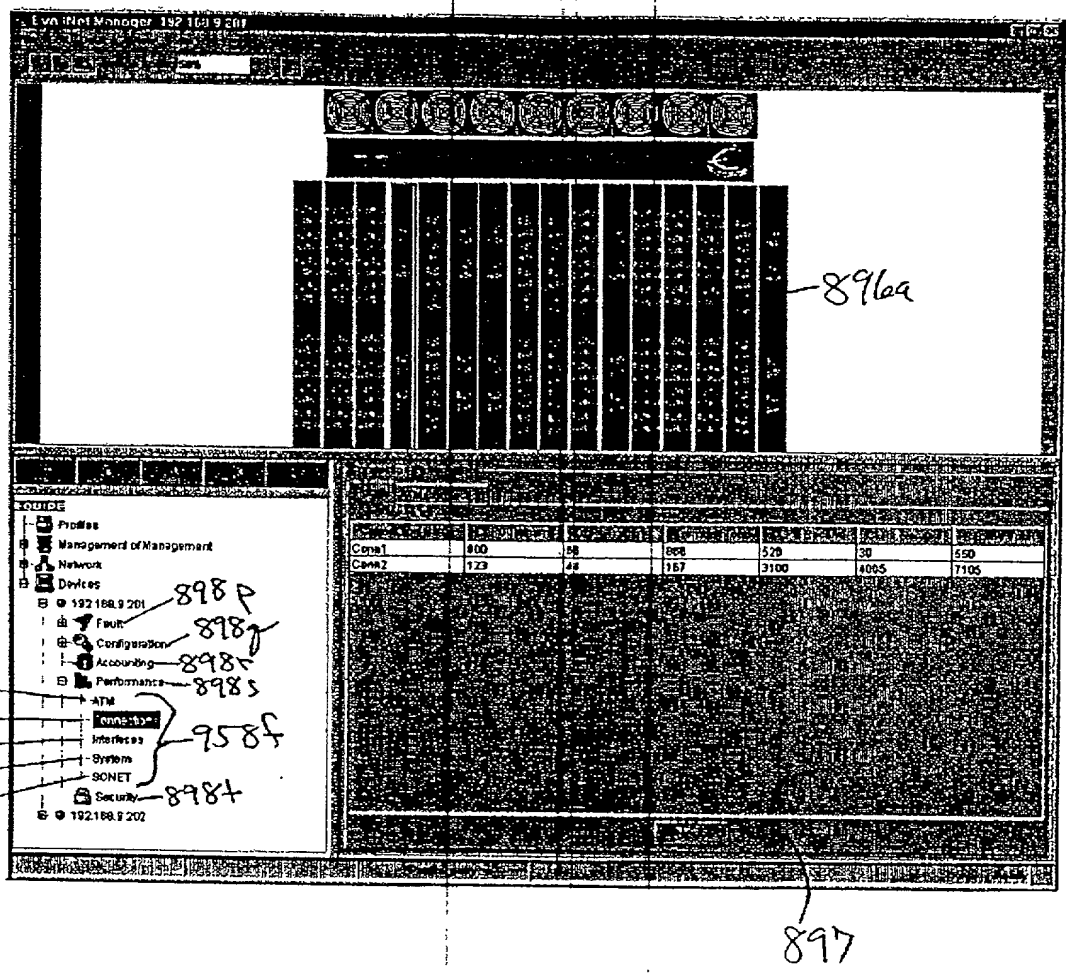

895
896a
898p
898q
898r
898s
958g
958h
958i
958j
958k
958f
898t
897

Fig. 7x

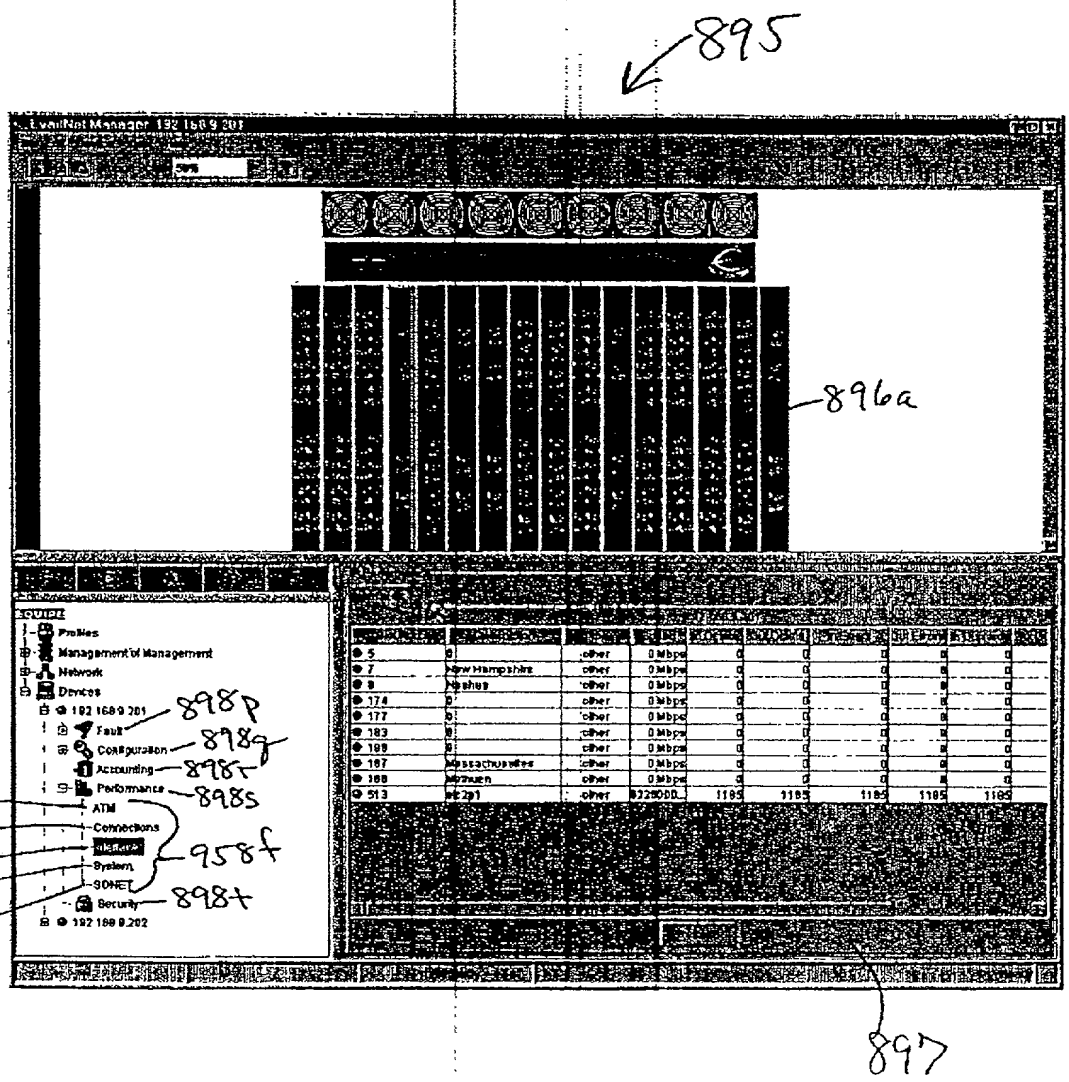

895
896a
897
898p
898q
898r
898s
898t
958g
958h
958i
958j
958k
958f
Probes
Management of Management
Network
Devices
Fault
Configuration
Accounting
Performance
ATM
Connections
Interface
System
SONET
Security

Fig. 7y

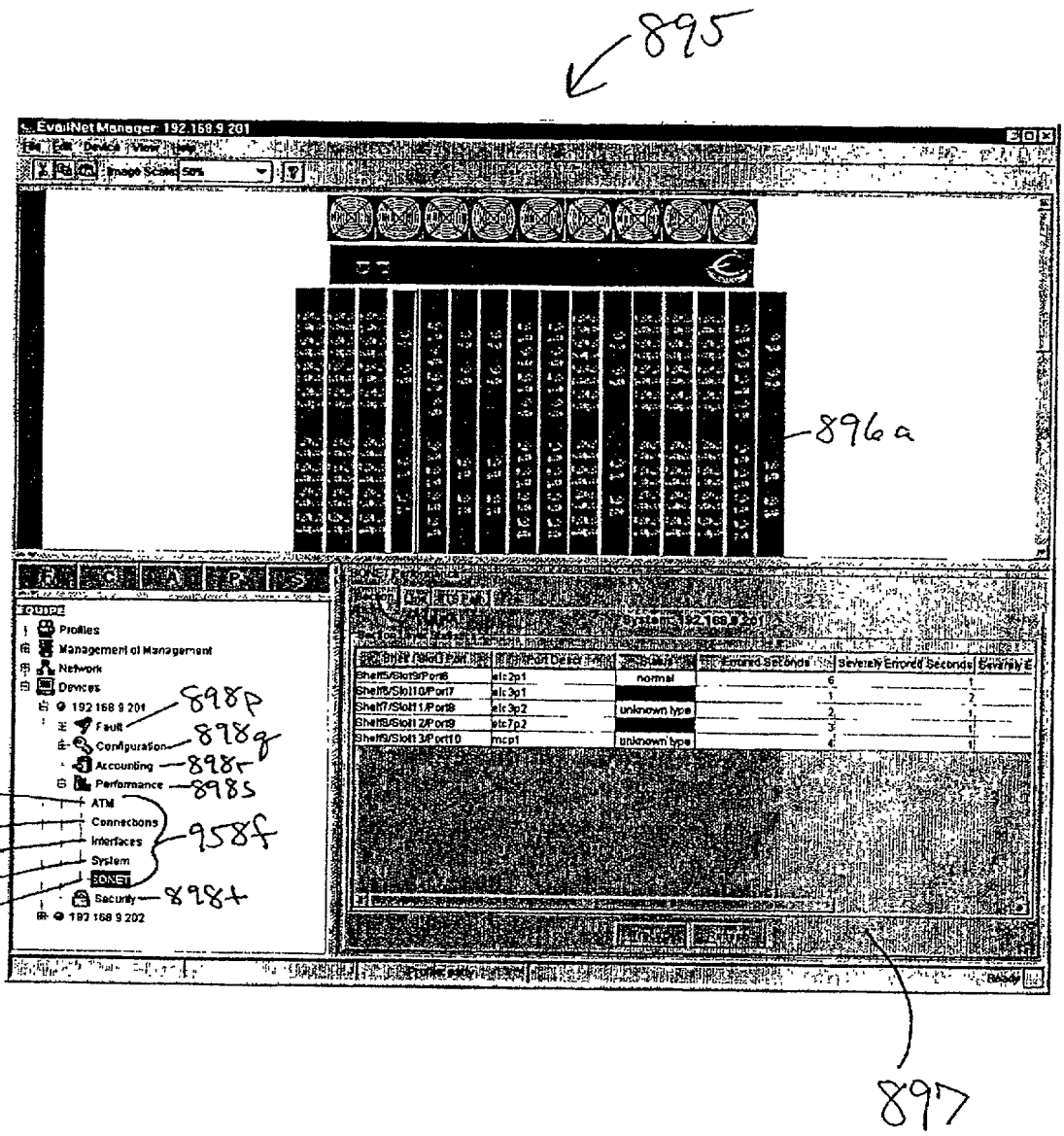

895
EquiNet Manager 192.168.9.201
896a
Profiles
Management of Management
Network
Devices
192.168.9.201
Fault
Configuration
Accounting
Performance
ATM
Connections
Interfaces
System
SONET
Security
192.168.9.202
898p
898q
898r
898s
898t
958g
958h
958i
958j
958k
958f
897

Fig. 8a

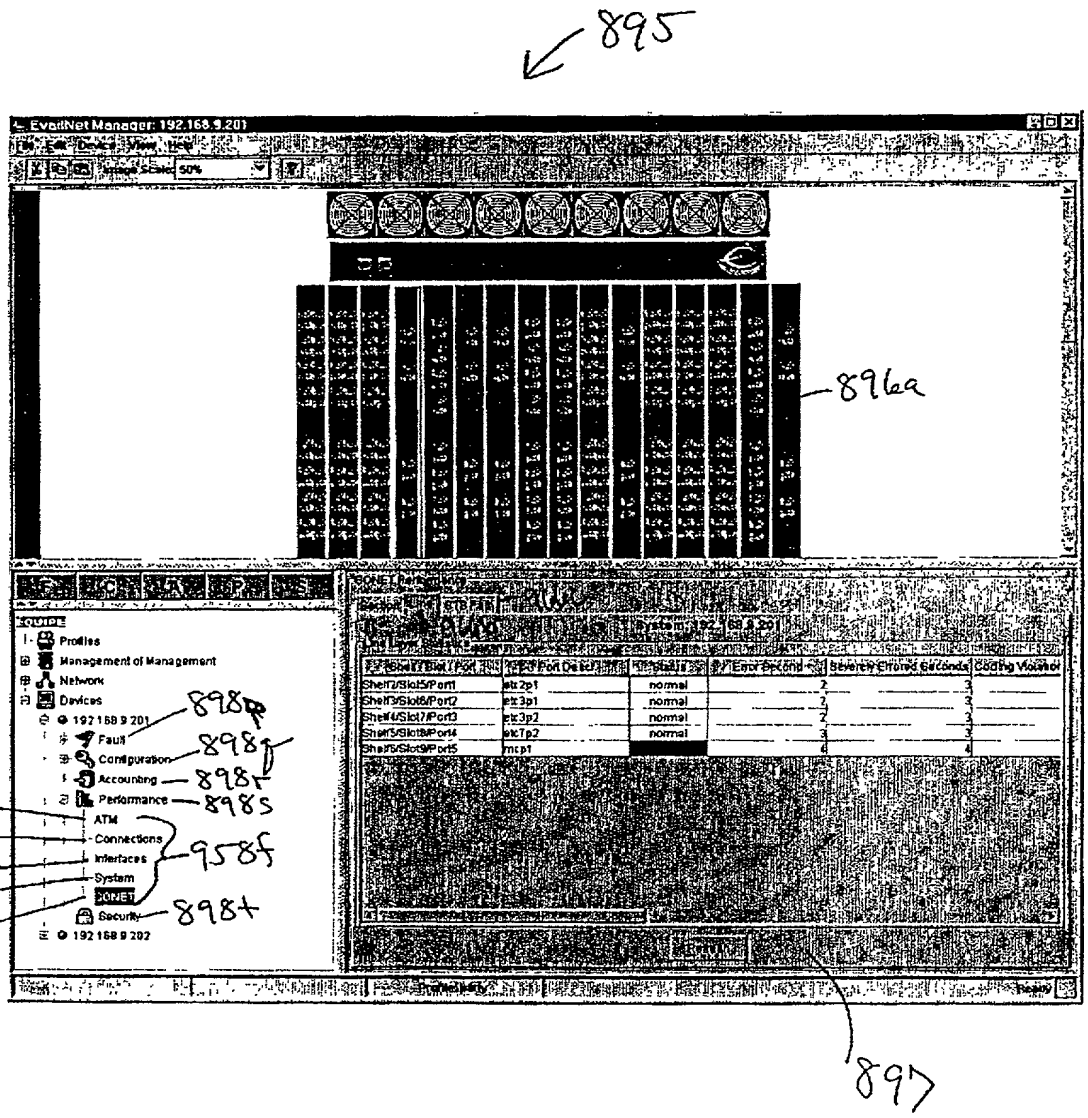

895
896a
897
898p
898q
898r
898s
898t
958g
958h
958i
958j
958k
958f
Profiles
Management of Management
Network
Devices
192.168.9.201
Fault
Configuration
Accounting
Performance
ATM
Connections
Interfaces
System
SONET
Security
192.168.9.202

Fig. 86

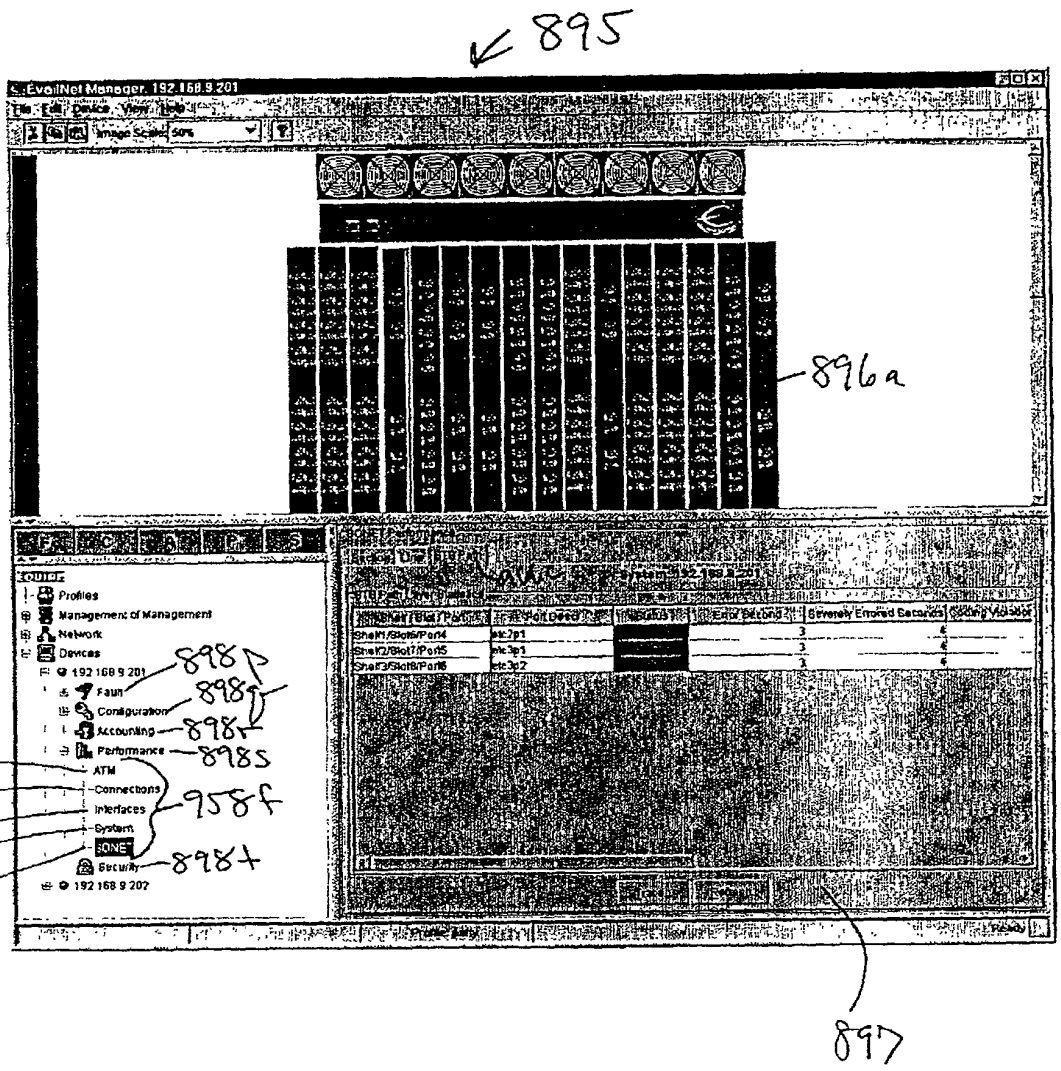

895
EvalNet Manager, 192.168.9.201
896a
Profiles
Management of Management
Network
Devices
192 168 9 201
Fault
Configuration
Accounting
Performance
ATM
Connections
Interfaces
System
SONET
Security
192 168 9 202
898
898g
898f
898t
898s
898f
898t
958g
958h
958i
958j
958k
897

Fig. 8c

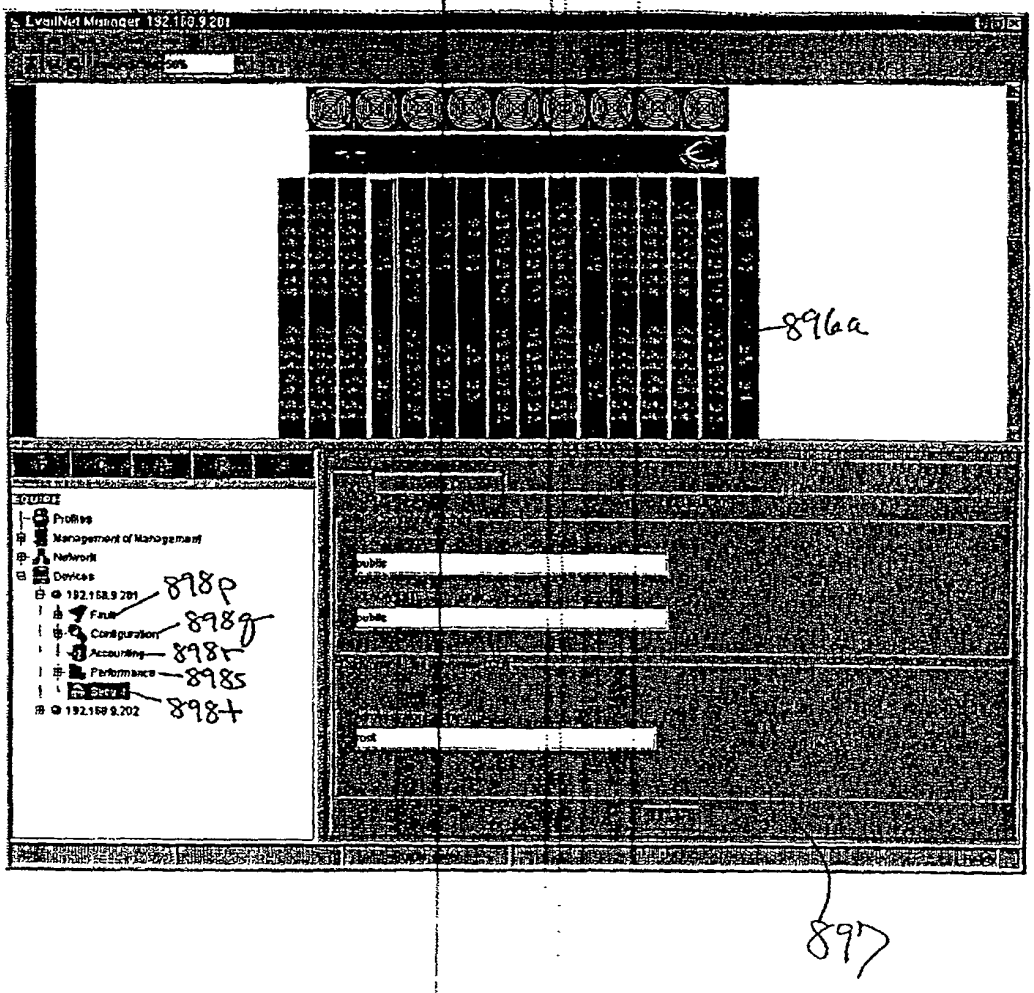

895
LevelNet Manager 192.168.9.201
896a
EQUIPE
Profiles
Management of Management
Network
Devices
192.168.9.201
Fault
Configuration
Accounting
Performance
Security
192.168.9.202
898p
898q
898r
898s
898t
897

Fig 8d

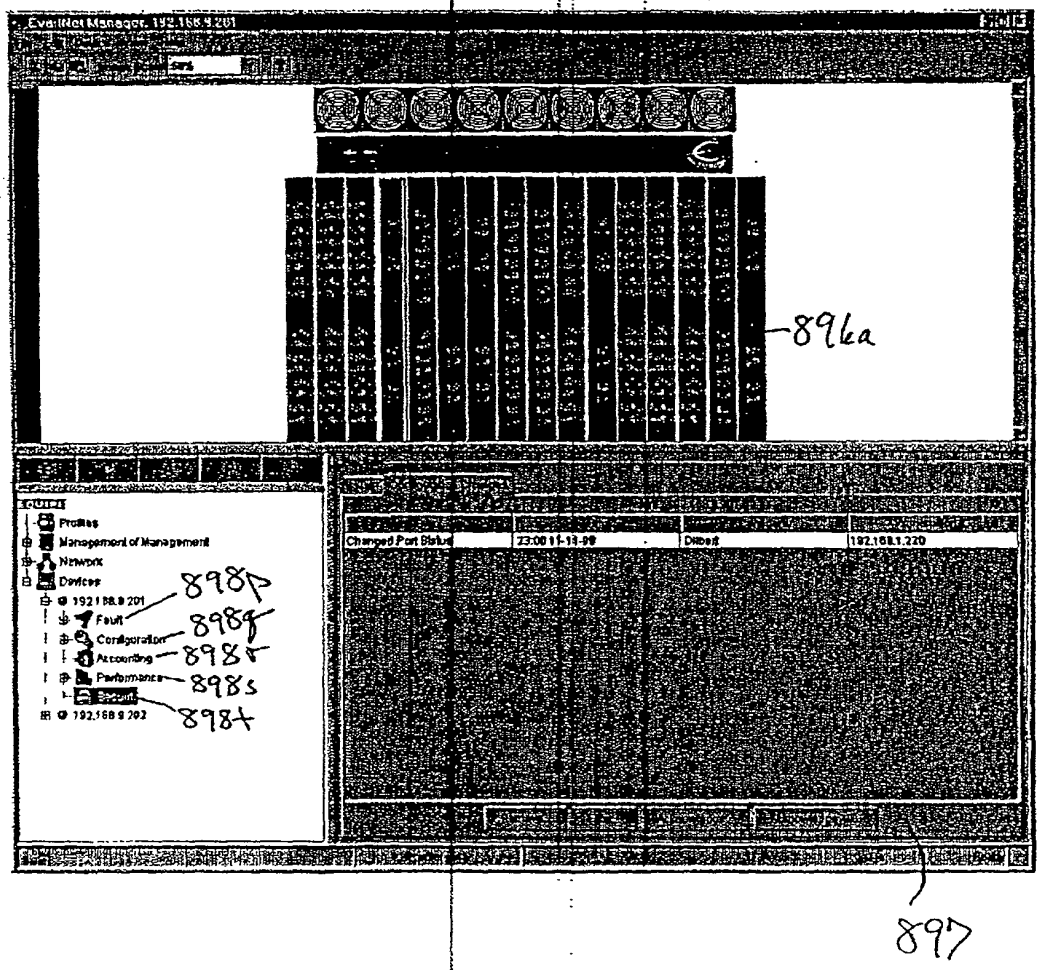

895
896a
Profiles
Management of Management
Network
Devices
Fault
Configuration
Accounting
Performance
Security
898p
898q
898r
898s
898t
897

Fig. 8e

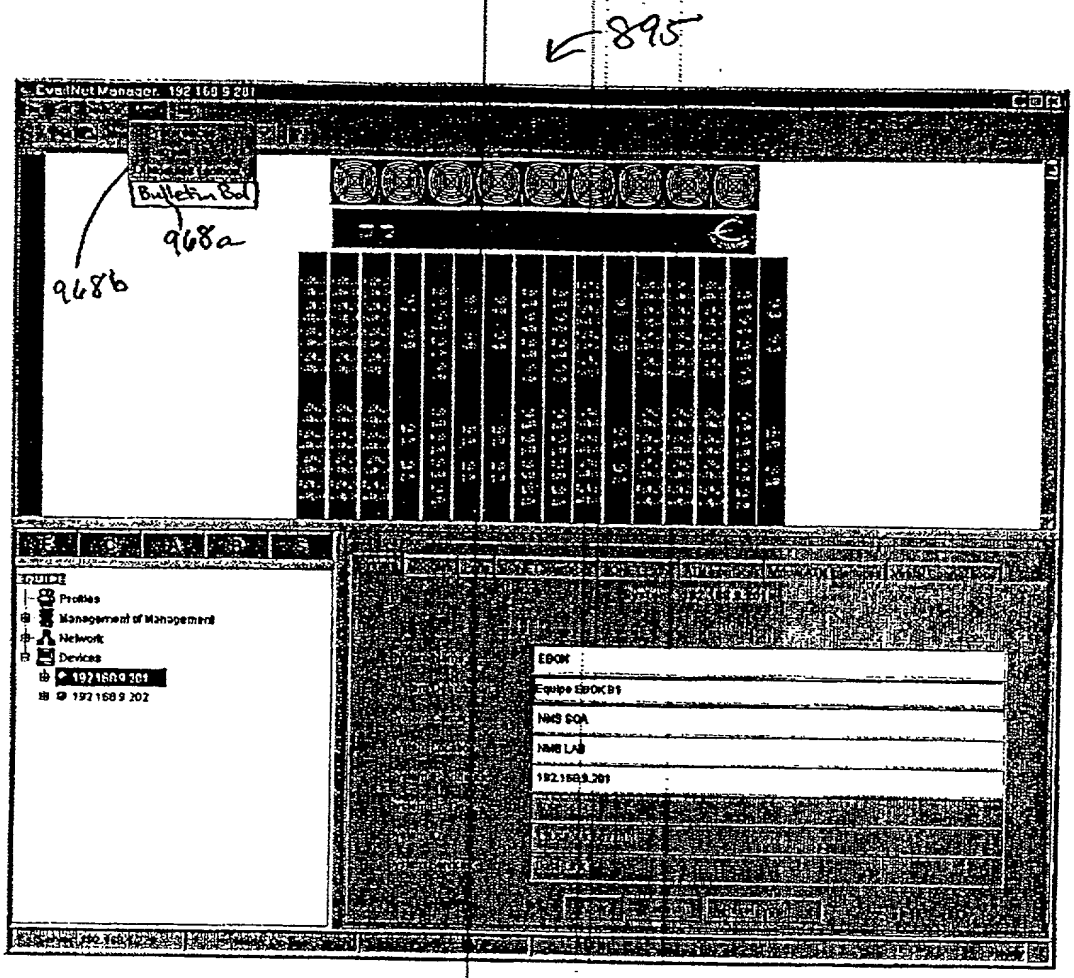

895
968a
968b
Profiles
Management of Management
Network
Devices
192.168.9.201
192.168.9.202
EDGN
Equipe EDGN
NMS SQA
NMS LAB
192.168.9.201

Fig. 9a

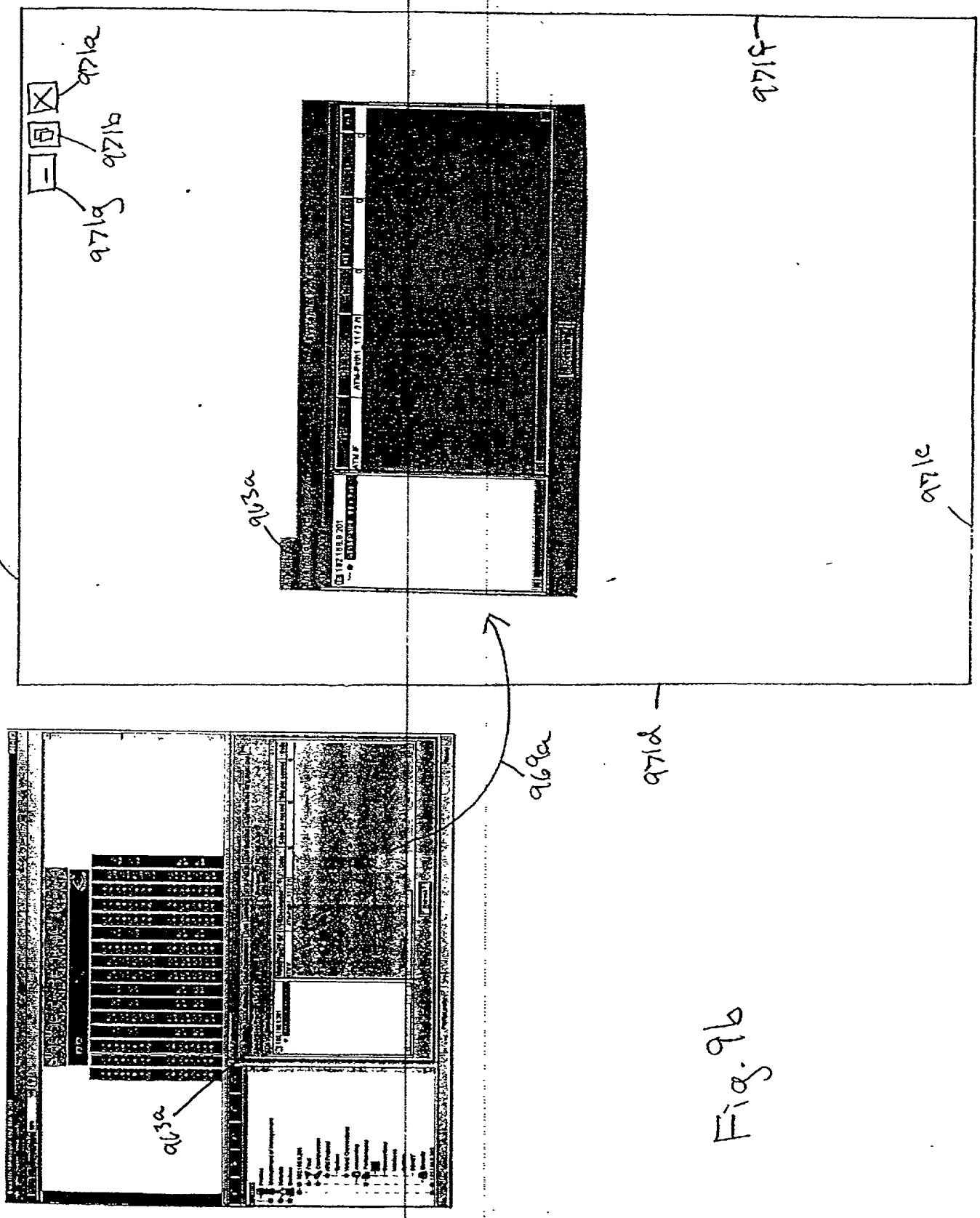


Fig. 9b

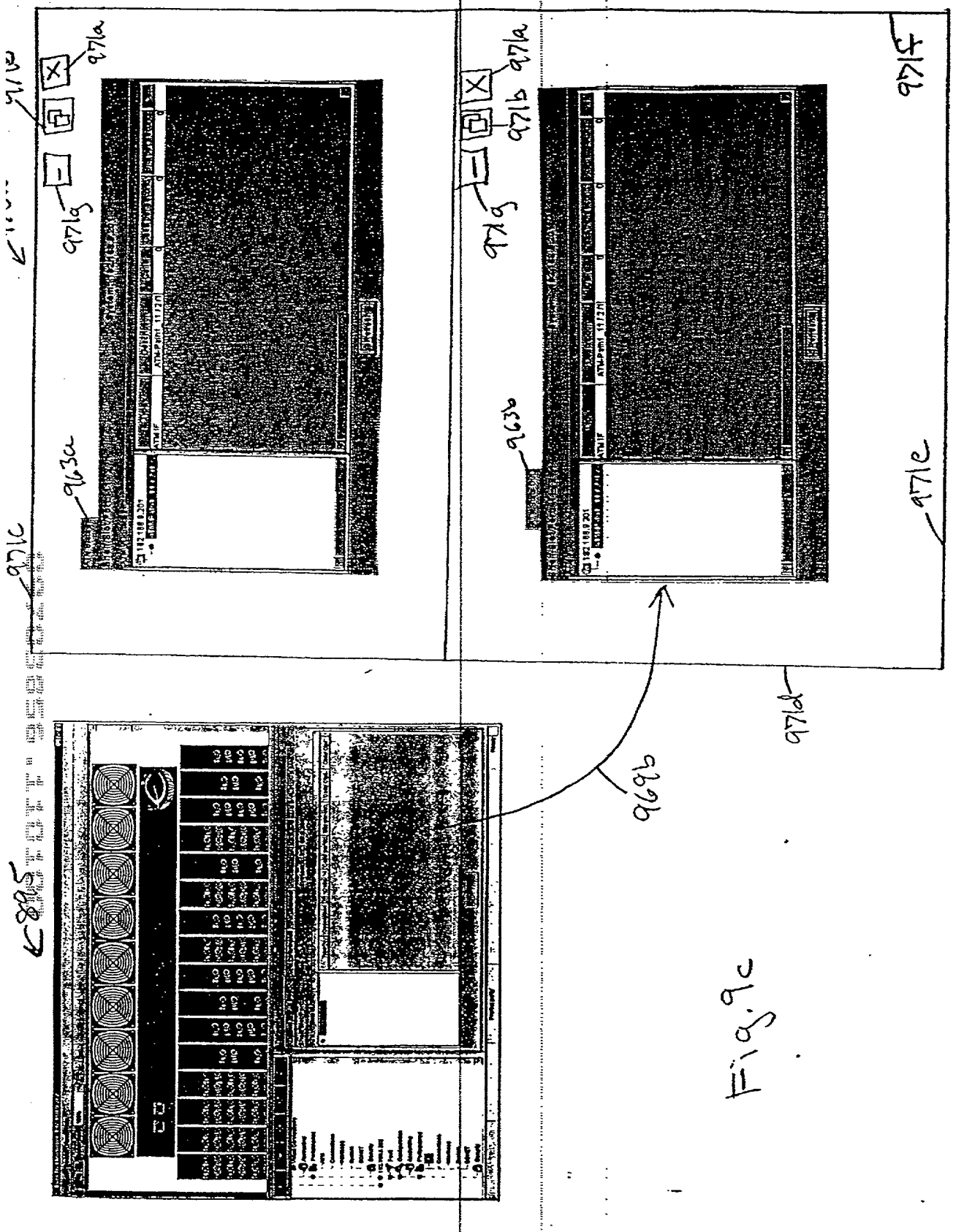


Fig. 9c

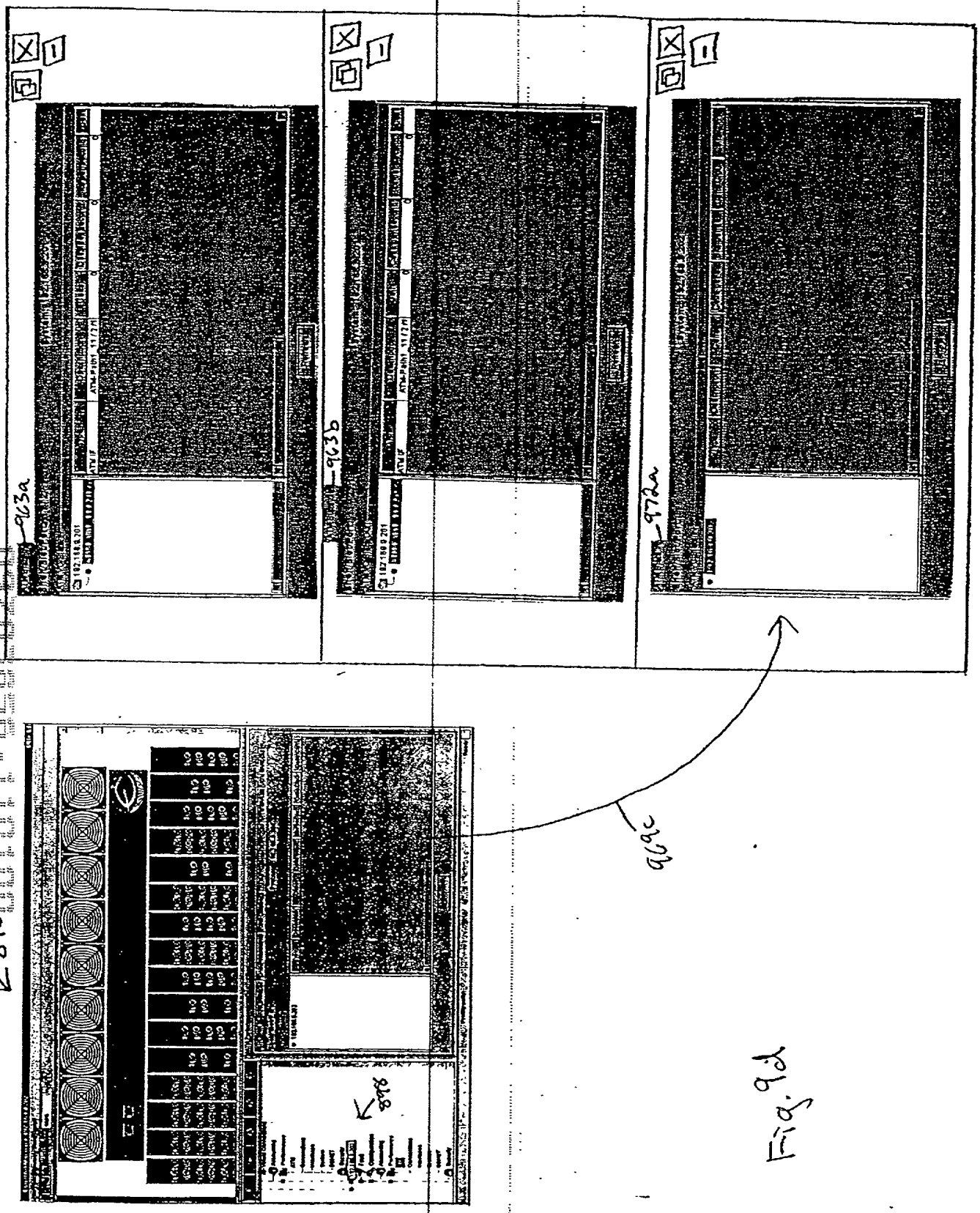


Fig. 9d

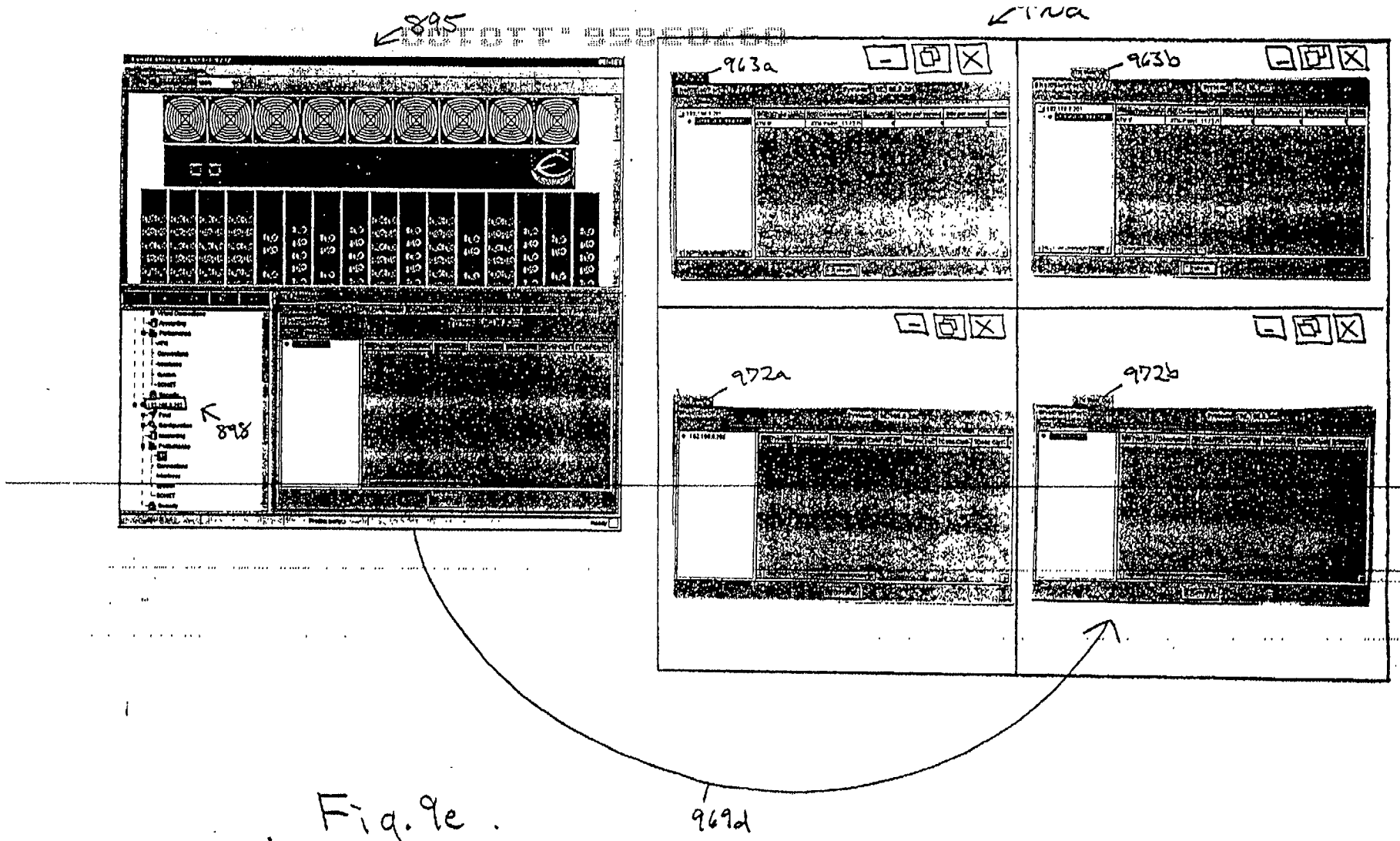


Fig. 9e.

Sep-5-00 1:42PM; Page 11/20

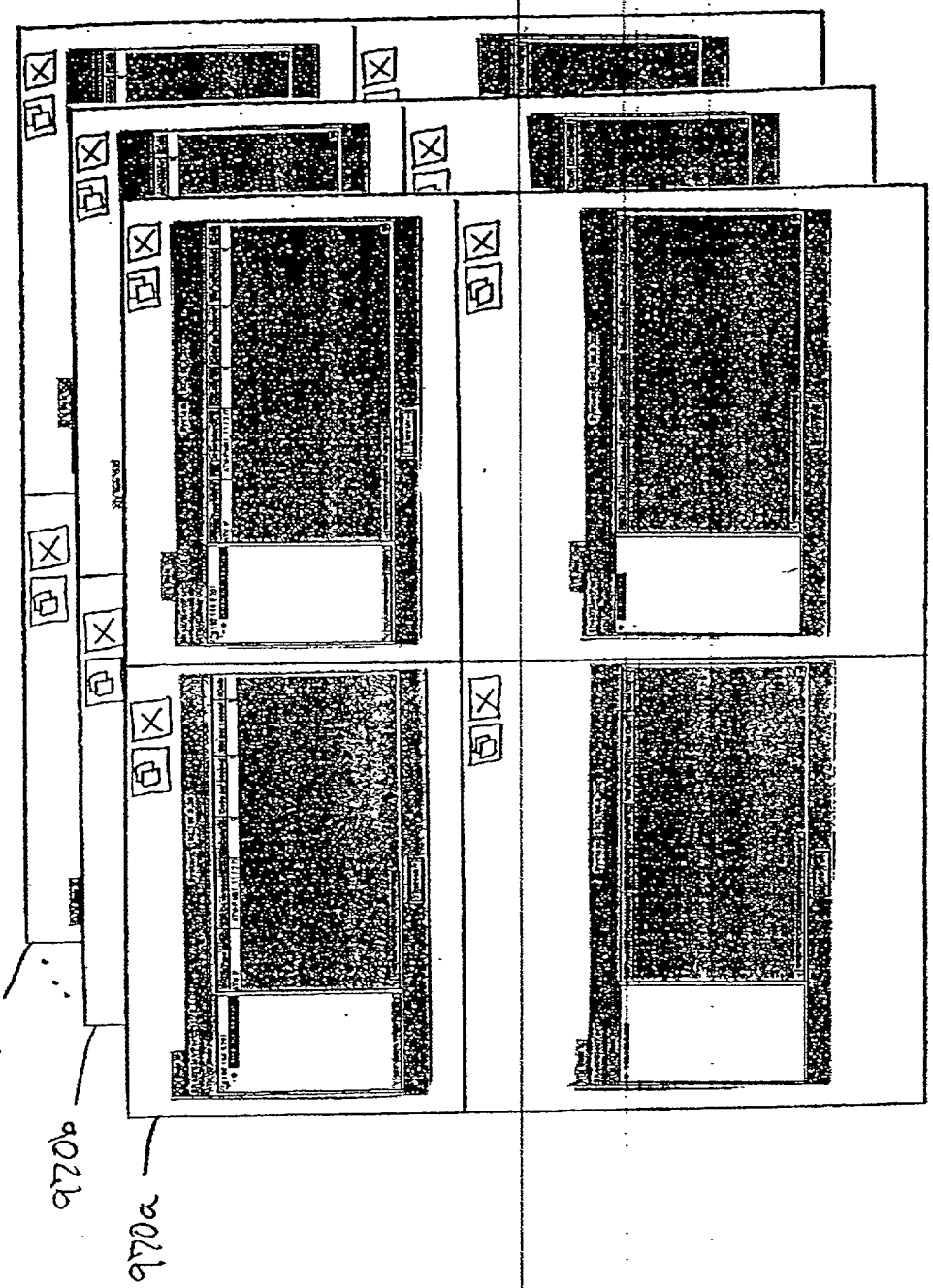


Fig. 9A

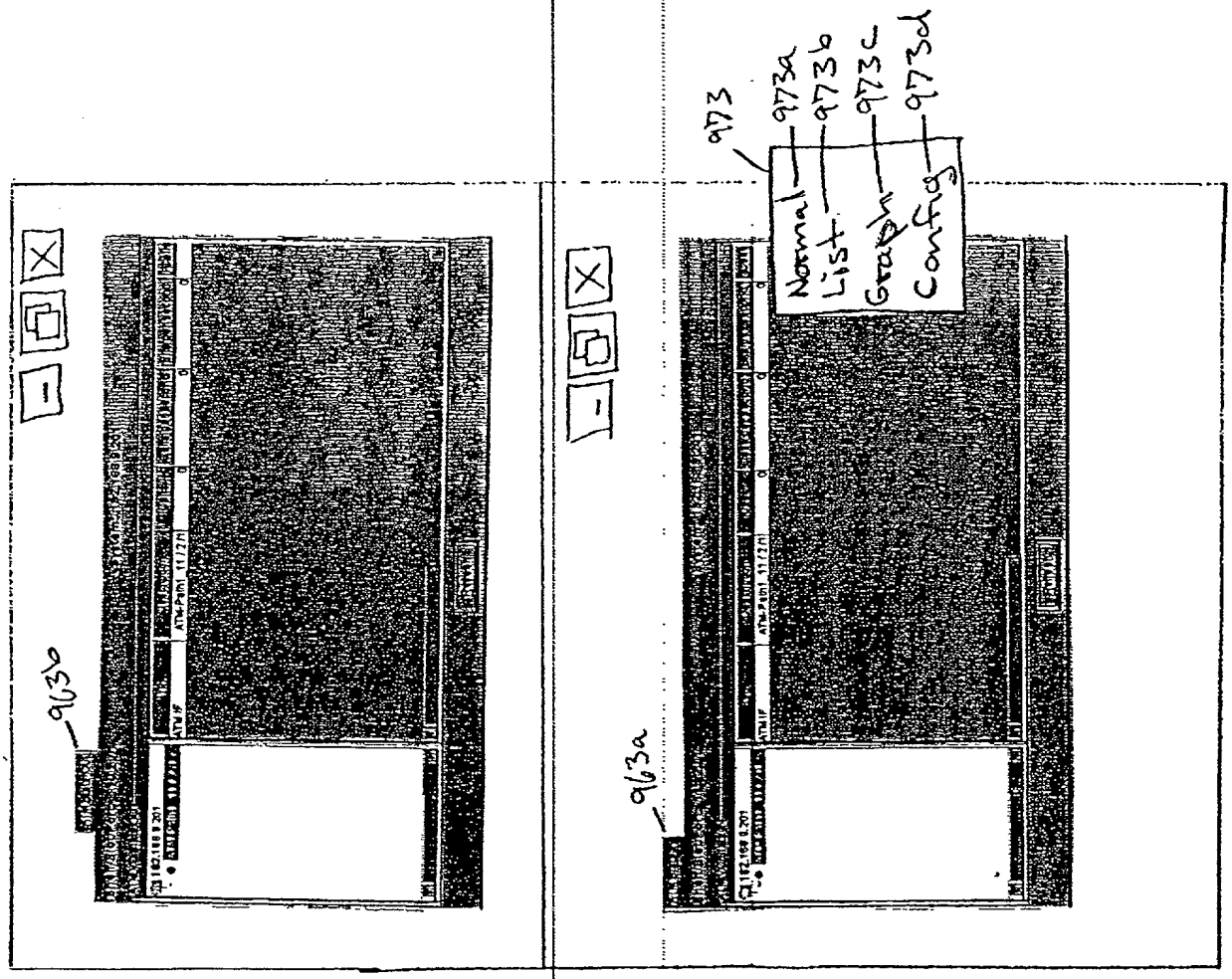

970a
963b
963a
973
973a
973b
973c
973d
Normal
List
Graph
Config

Fig. 9g

Sent By: Equipe Communications; 781 795 2031; Sep-5-00 1:46PM; Page 14/20

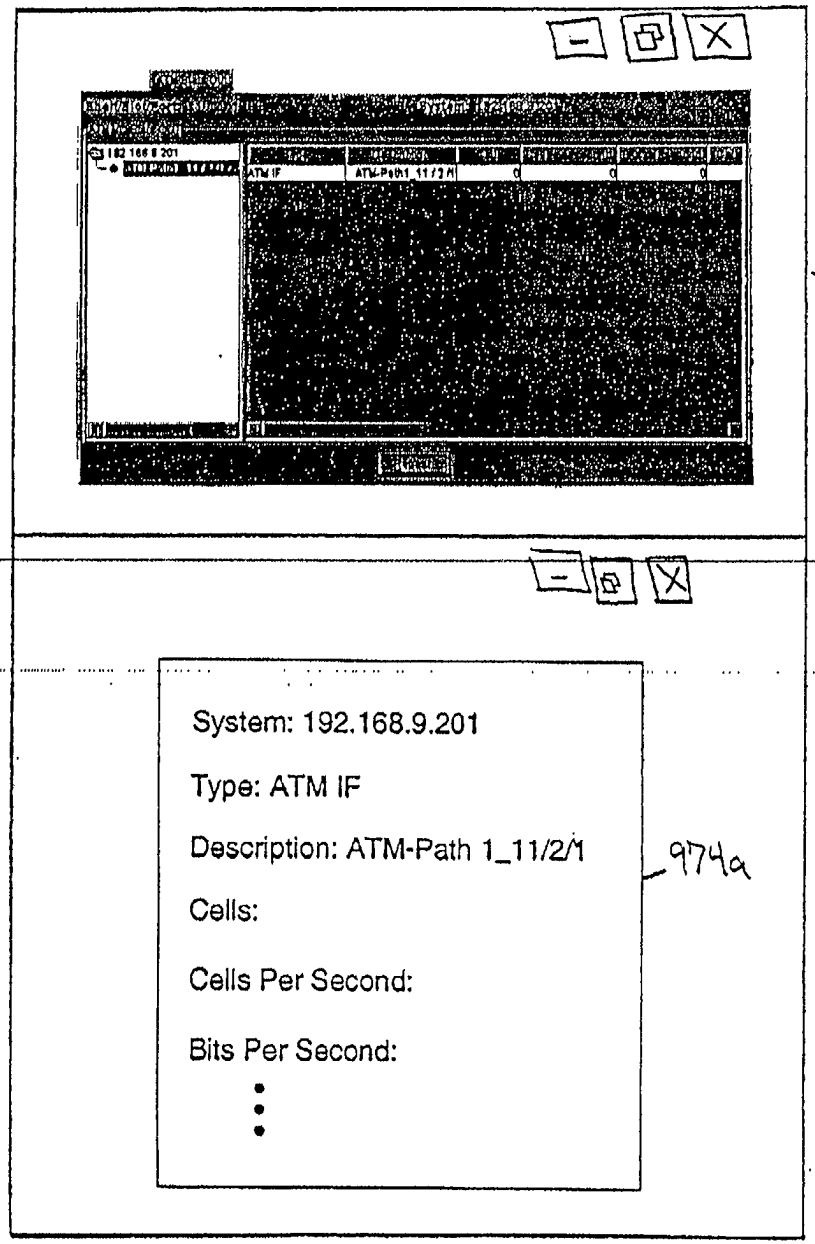


Fig. 9h

Sent By: Equipe Communications; 781 795 2031; Sep-5-00 1:46PM; Page 15/20

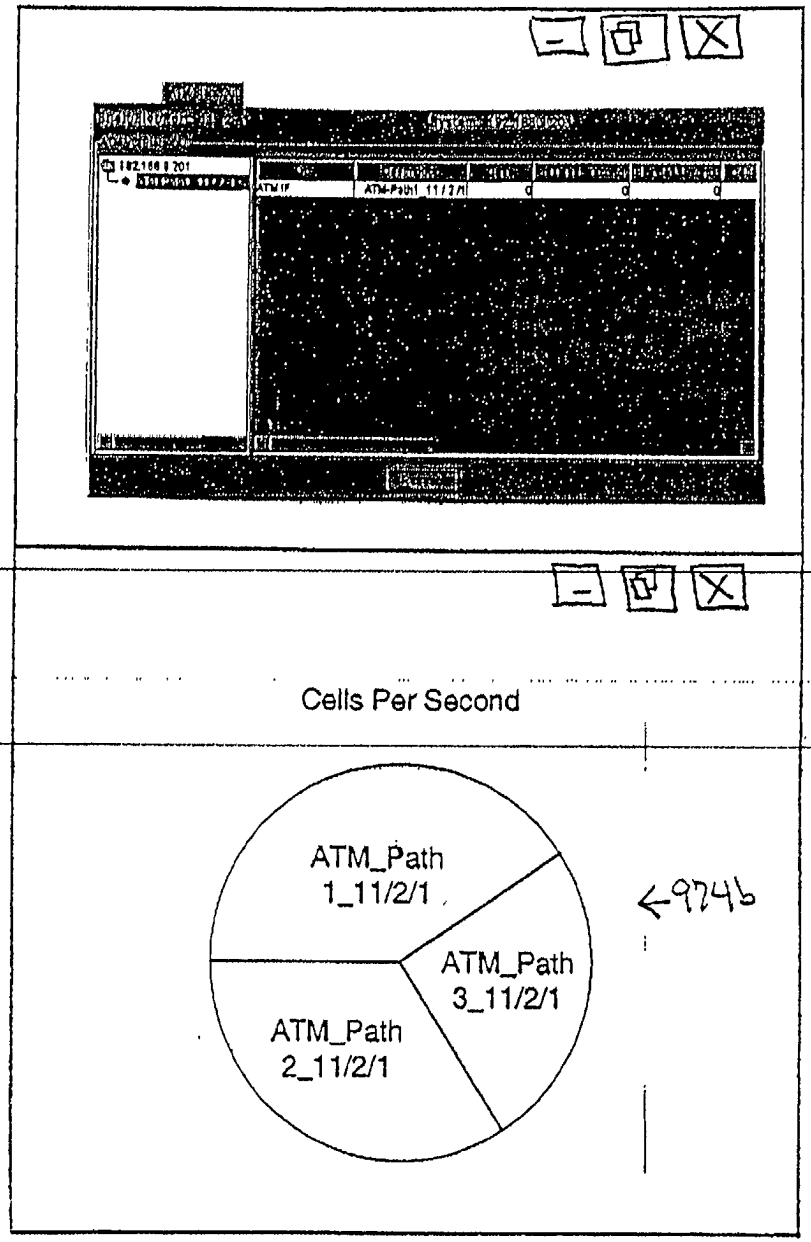


Fig. 9

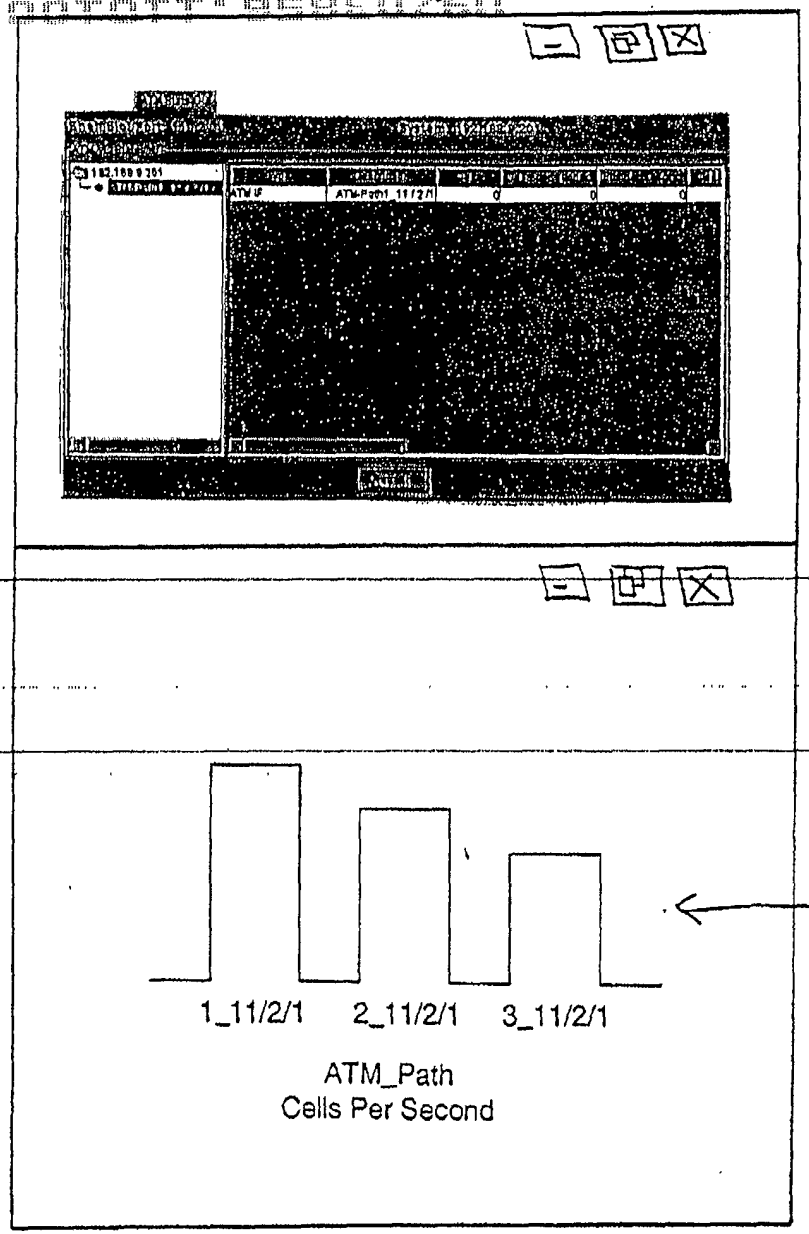
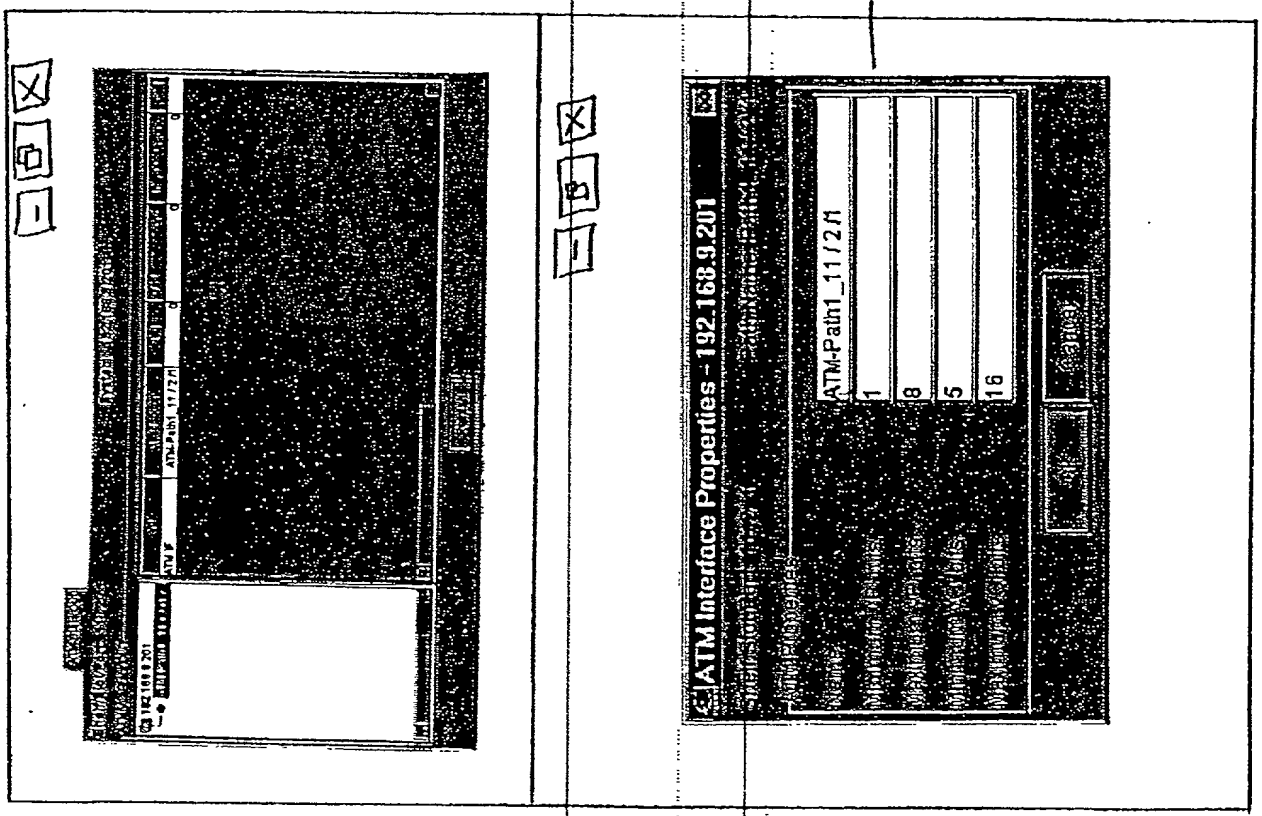


Fig. 9J


ATM Interface Properties - 192.168.9.201
ATM-Path1_1112/1
1
8
5
16

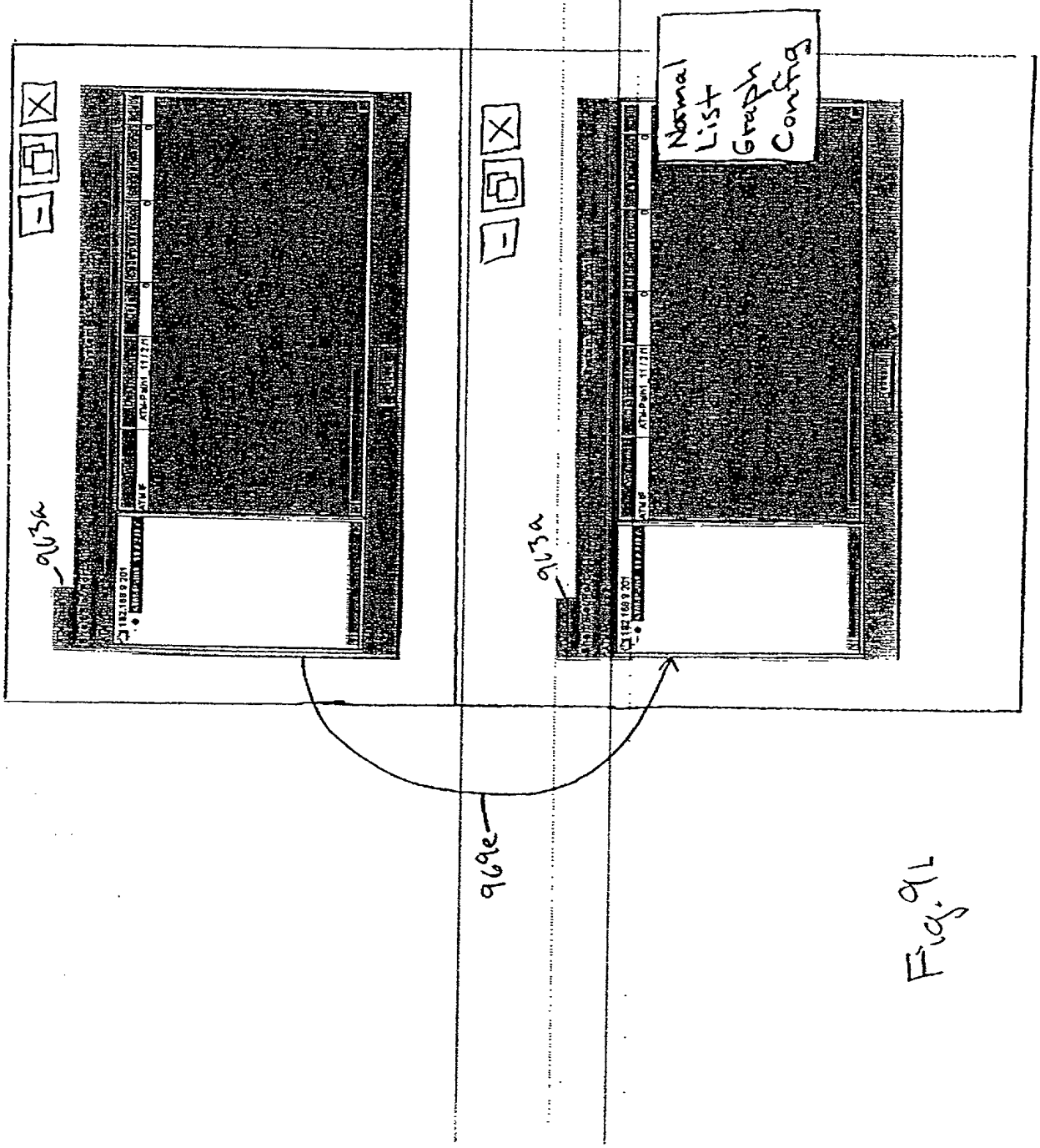


Fig. 9L

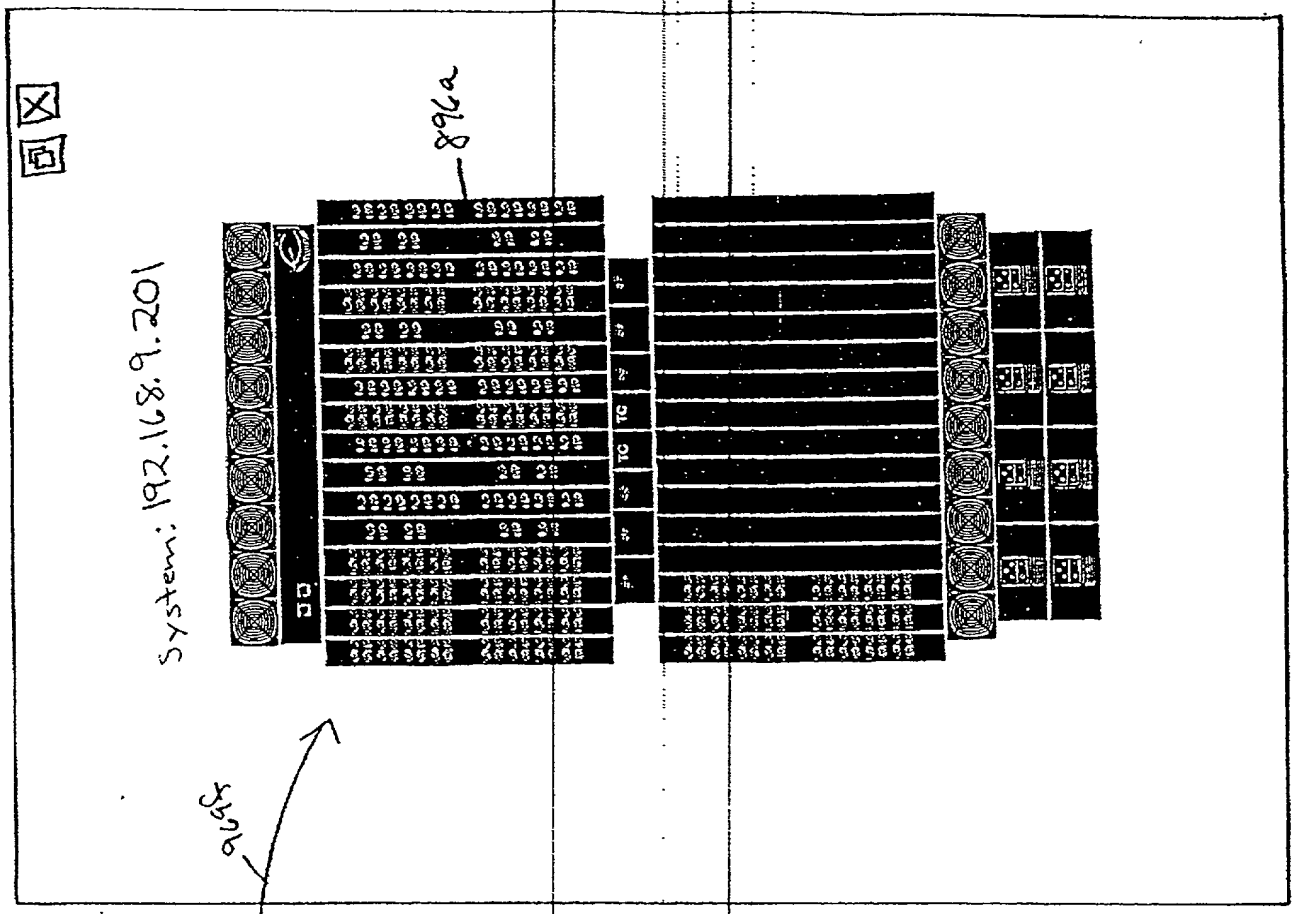


Fig. 9m

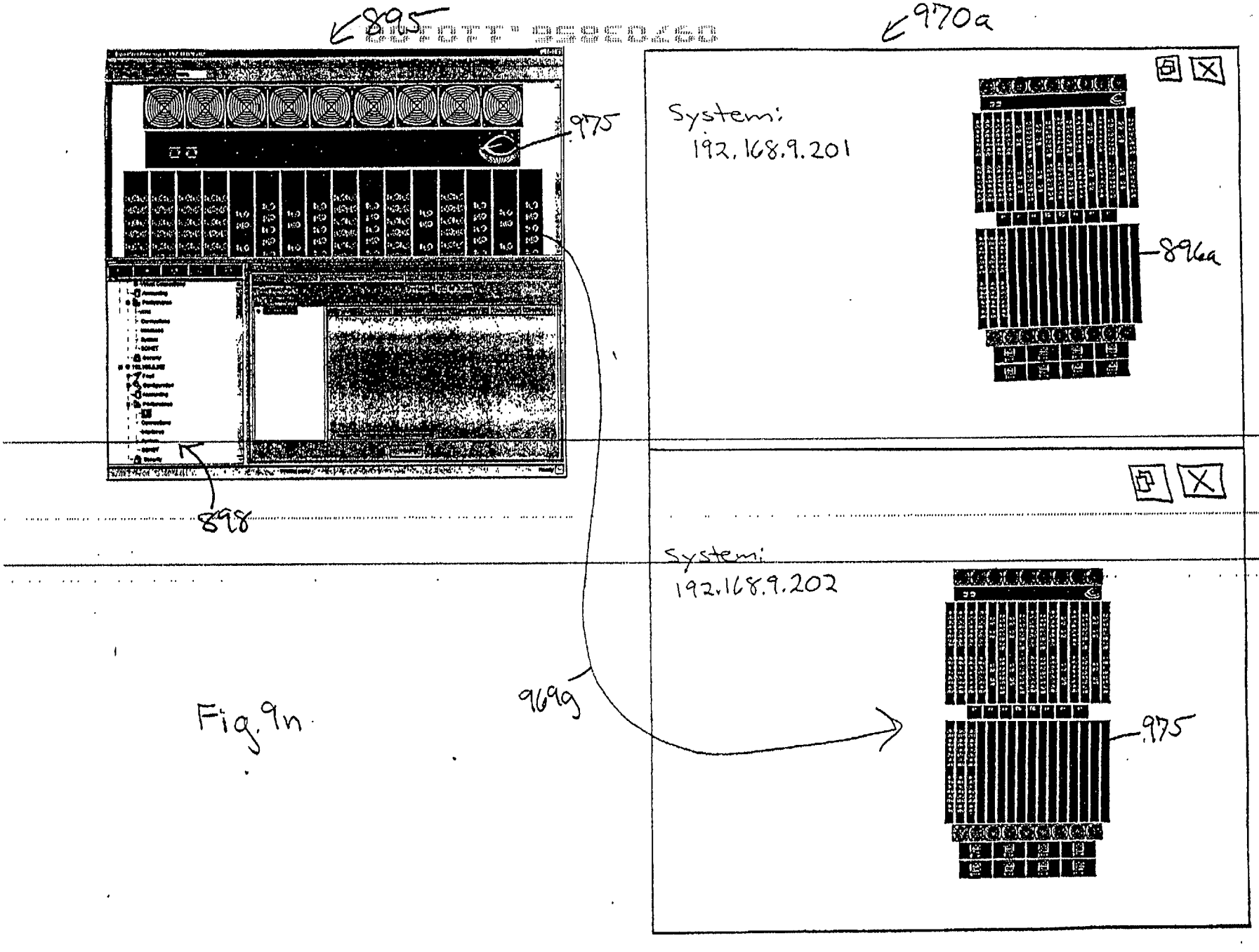


Fig. 9n

ent By: Equipe Communications; 781 795 2031; Sep-5-00 1:49PM; Page 20/20

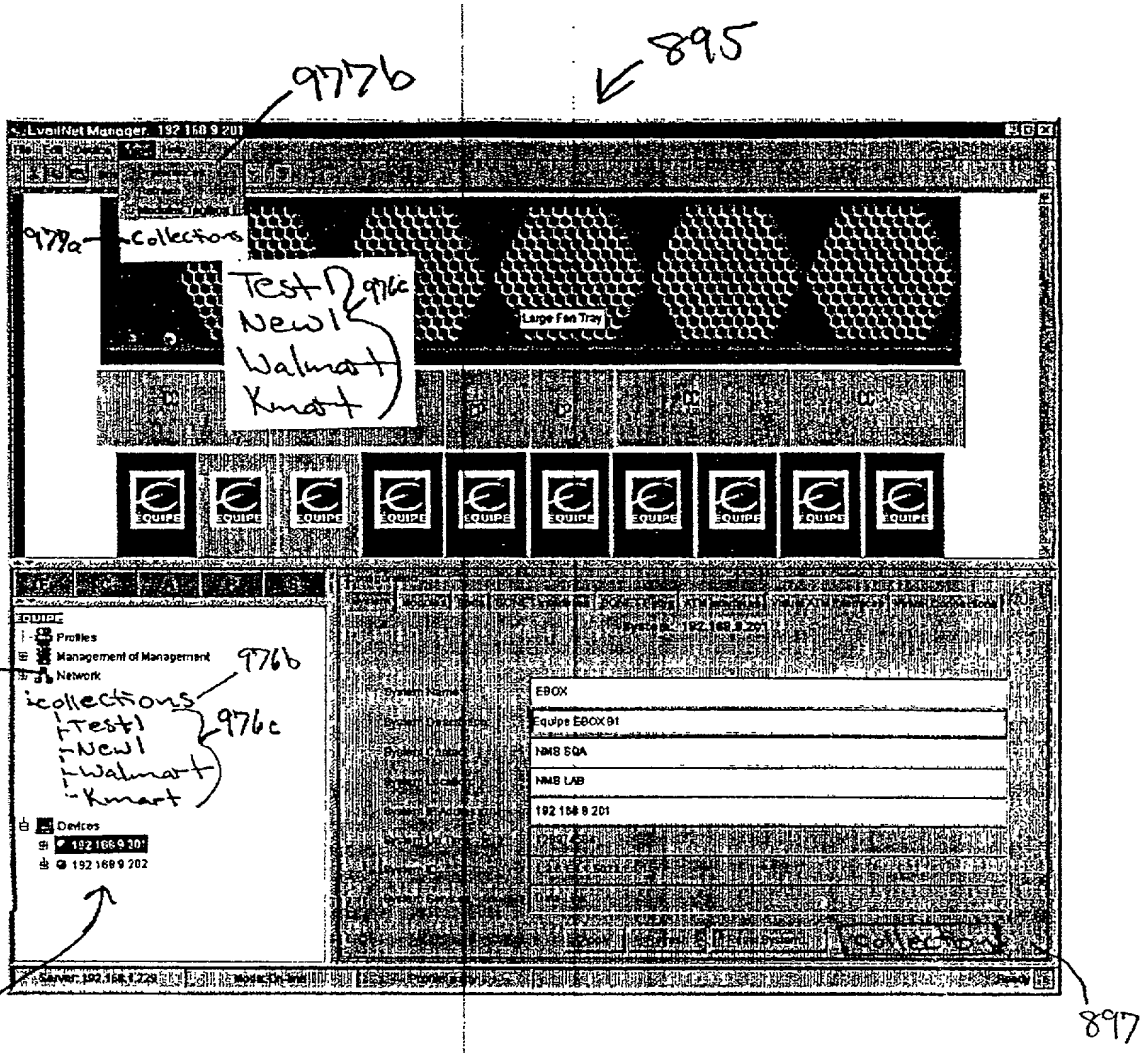

895
976b
976a
Collections
Test1
New1
Walmart
Kmart
976c
Large Fan Tray
Profiles
Management of Management
Network
Collections
Test1
New1
Walmart
Kmart
Devices
EBOX
Equipe EBOX 91
NMS SQA
NMS LAB
192 168 9 201
898
897

Fig. 10a

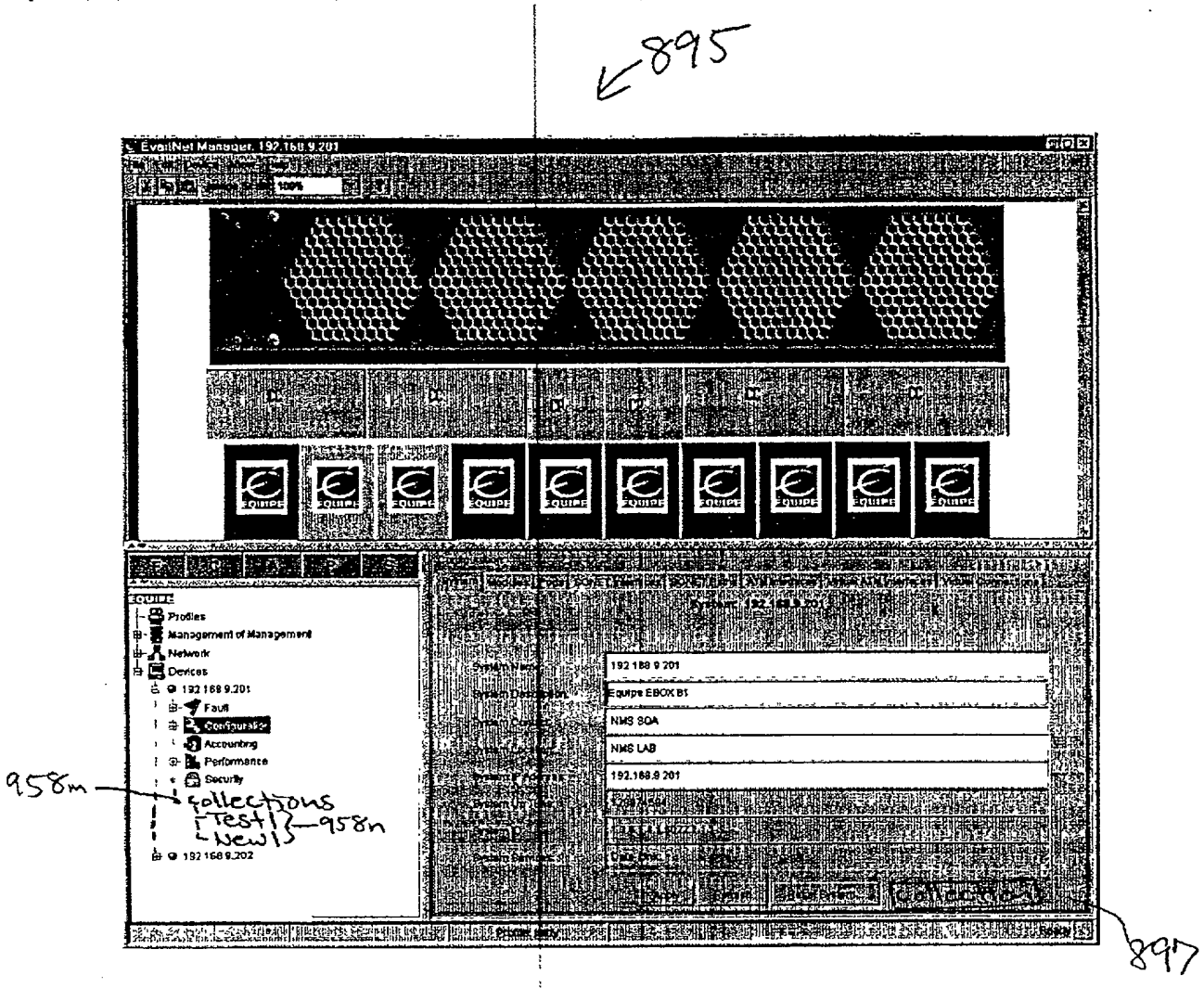

895
EventNet Manager, 192.168.9.201
EQUIPE
Profiles
Management of Management
Network
Devices
192.168.9.201
Fault
Configuration
Accounting
Performance
Security
958m
collections
Test1
New15
958n
192.168.9.202
192.168.9.201
Equipe EBOX B1
NMS SOA
NMS LAB
192.168.9.201
897

Fig. 10b

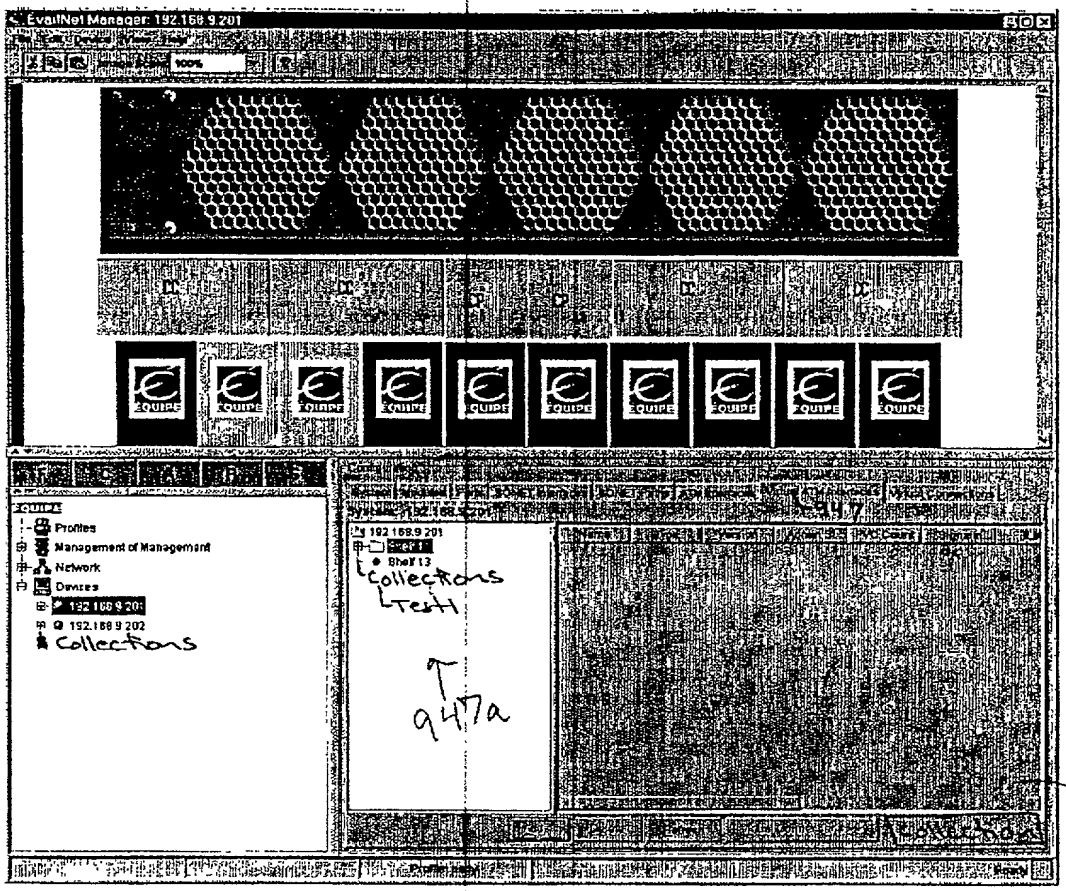

EvaliNet Manager: 192.168.9.201
Profiles
Management of Management
Network
Devices
192.168.9.201
192.168.9.202
Collections
947
192.168.9.201
Shelf 13
Collections
Test1
947a
897

Fig. 10c

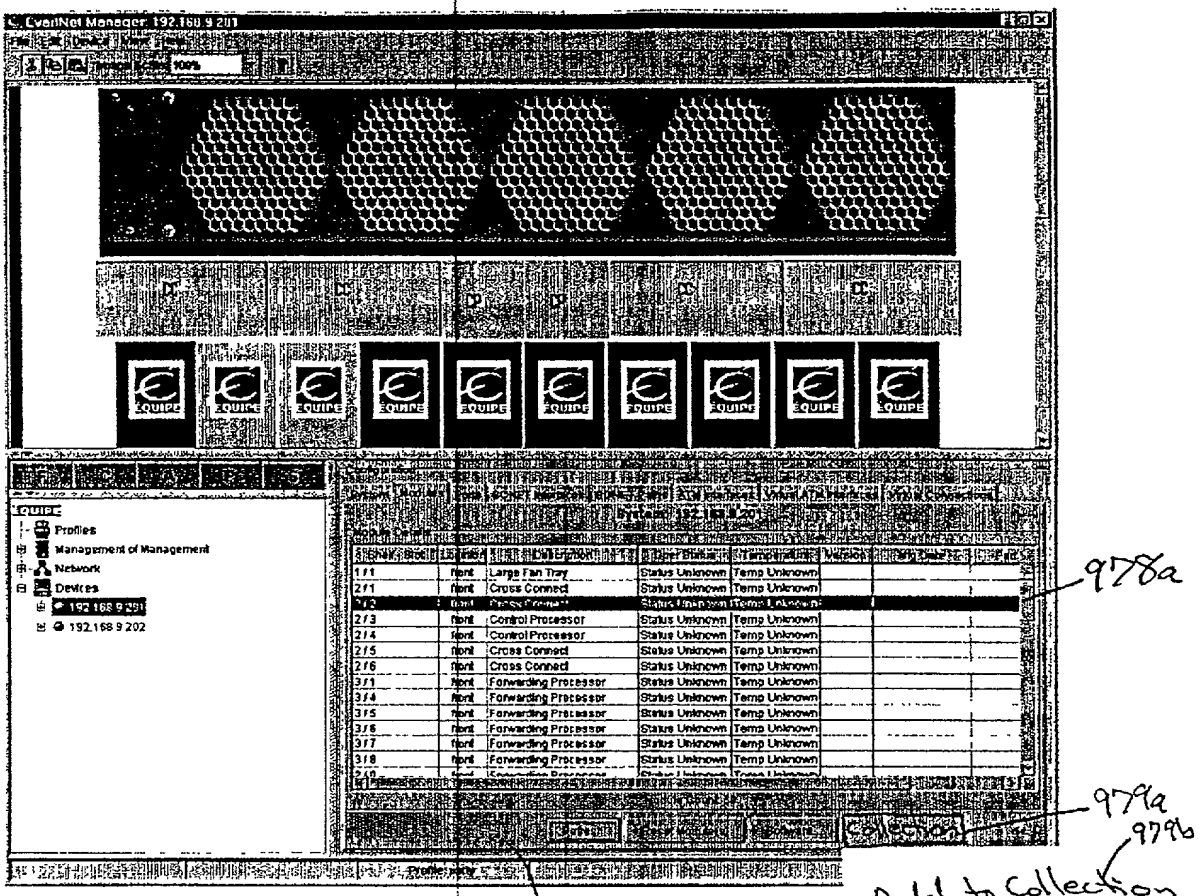


Fig. 10d

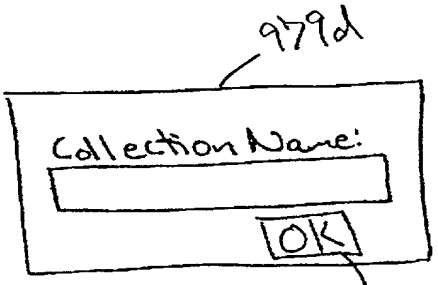


Fig. 10e

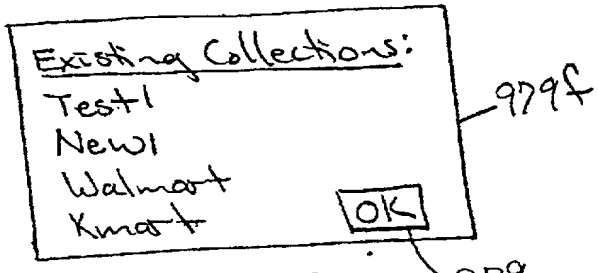


Fig. 10f

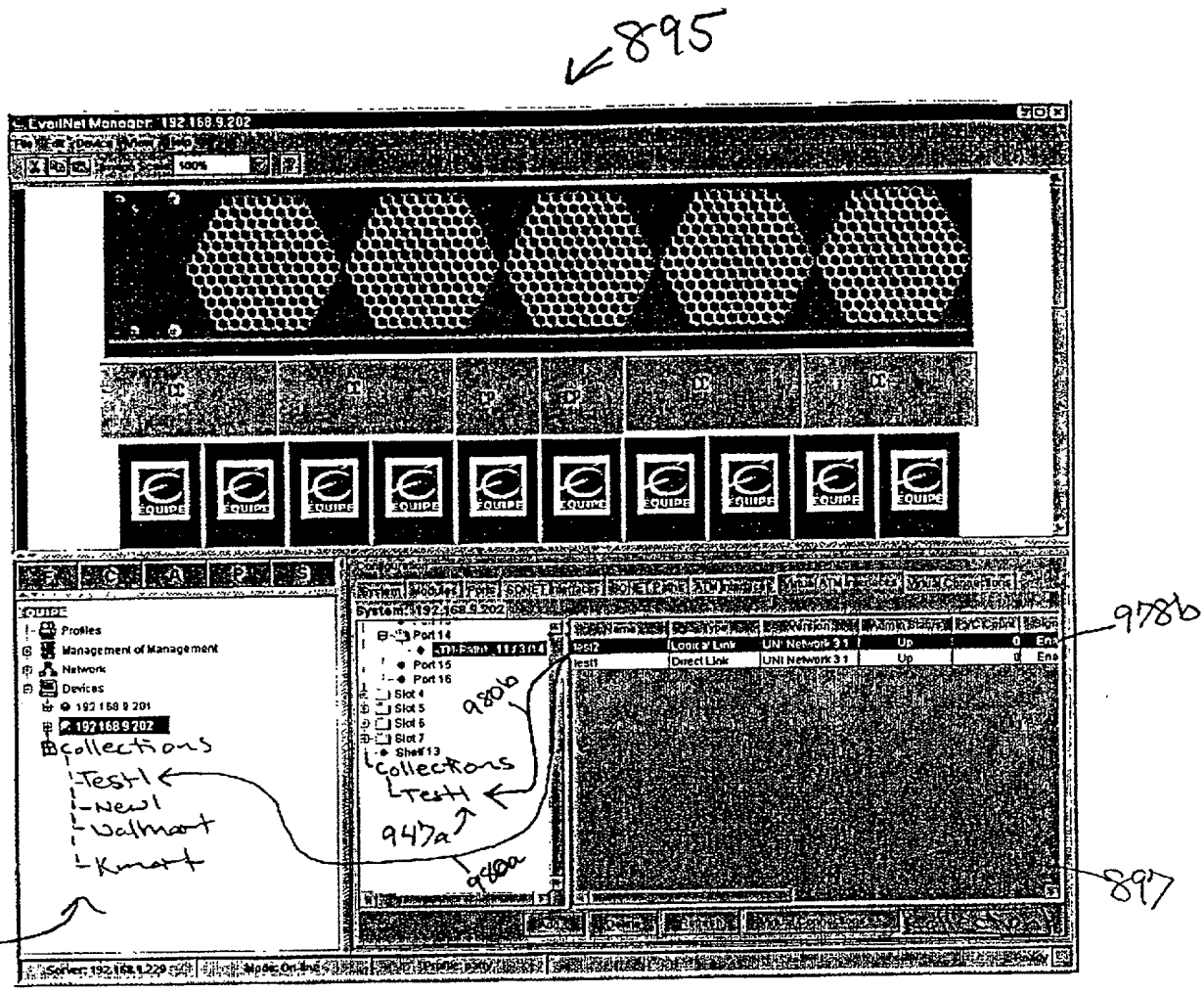

895
EvoliNet Manager 192.168.9.202
Profiles
Management of Management
Network
Devices
192.168.9.201
192.168.9.202
collections
Test1
New1
Walmart
Kmart
898
Port 14
Port 15
Port 16
Slot 4
Slot 5
Slot 6
Slot 7
Shelf 13
collections
Test1
947a
980a
980b
978b
897

Fig. 10g

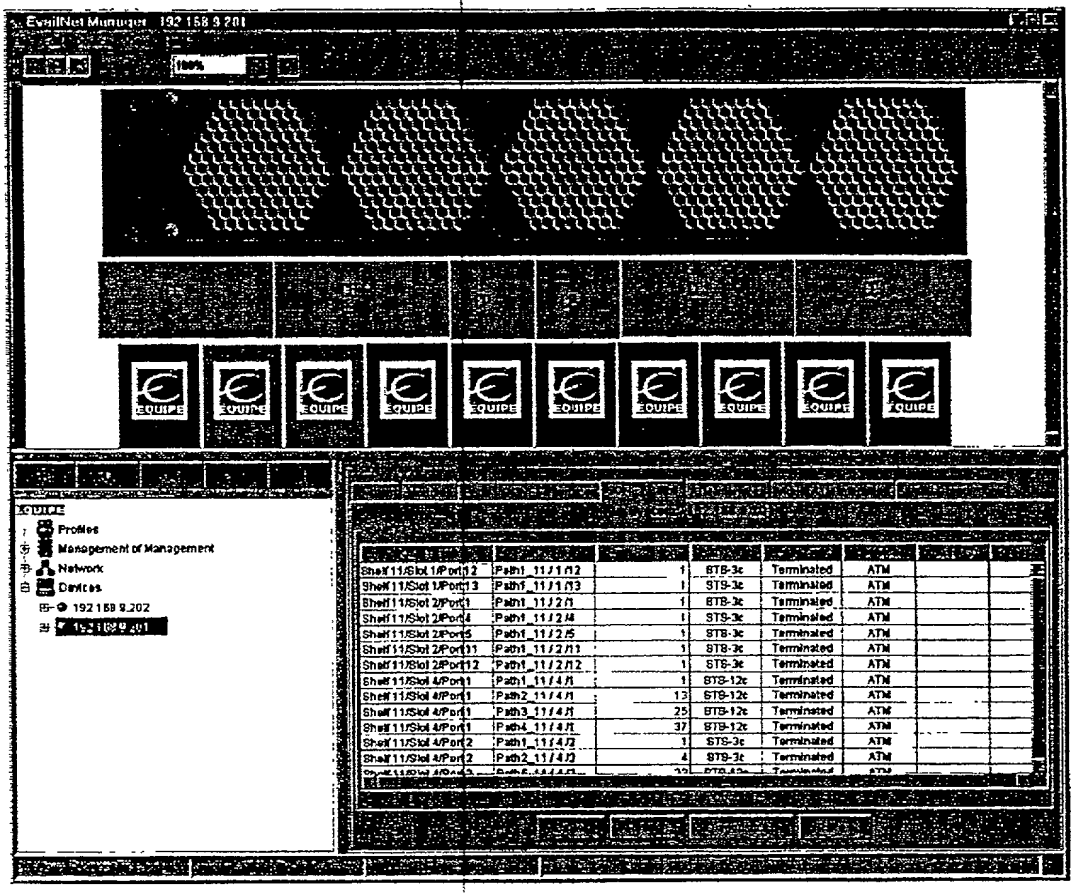

EQUIPE
Profiles
Management of Management
Network
Devices

Fig. 10h

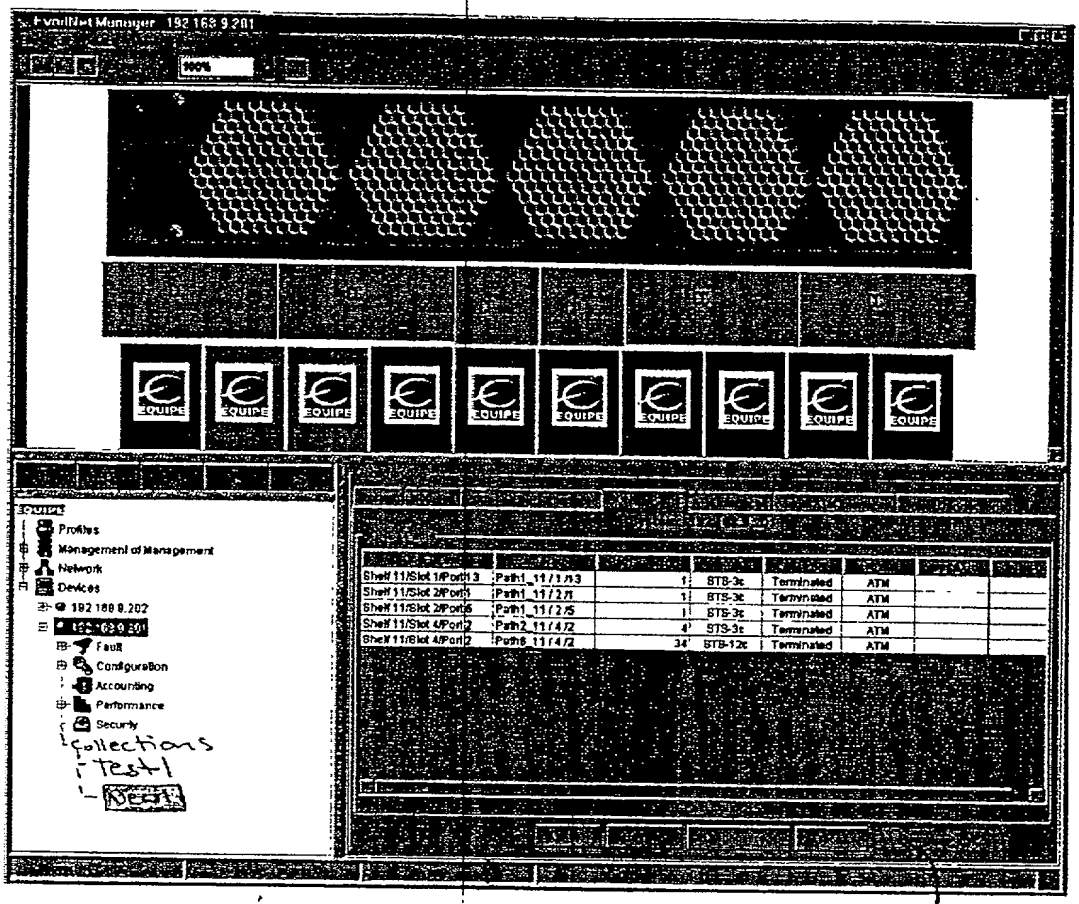

EQUIPE
Profiles
Management of Management
Network
Devices
Fault
Configuration
Accounting
Performance
Security
collections
test1
Next
982

Fig. 10i

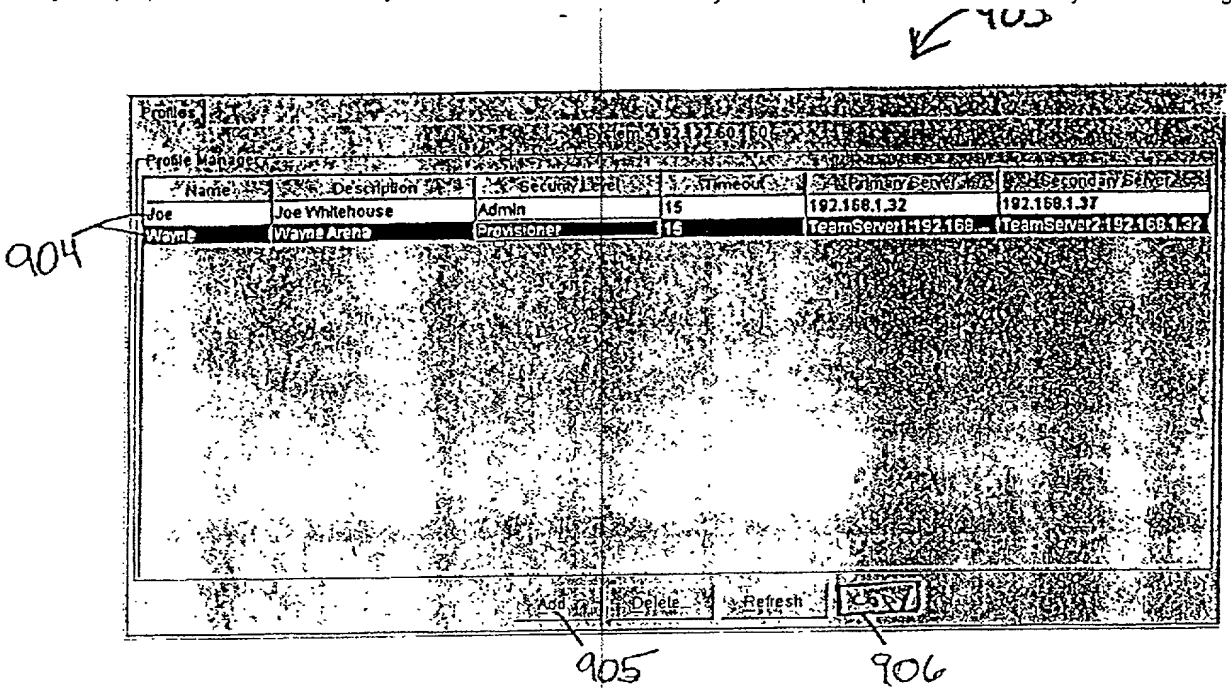
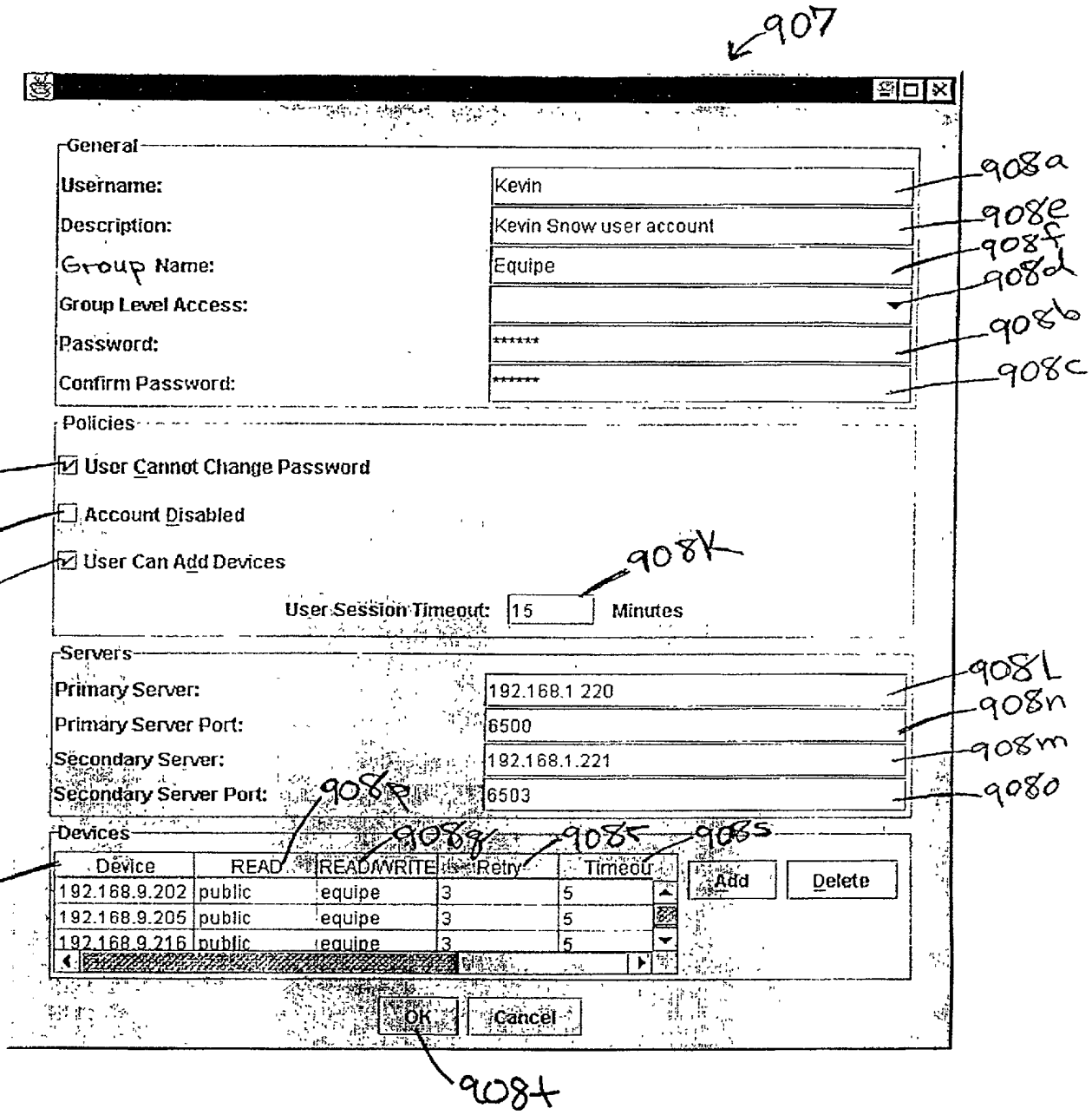


Fig. 11a

DRAFT - 09/15/00

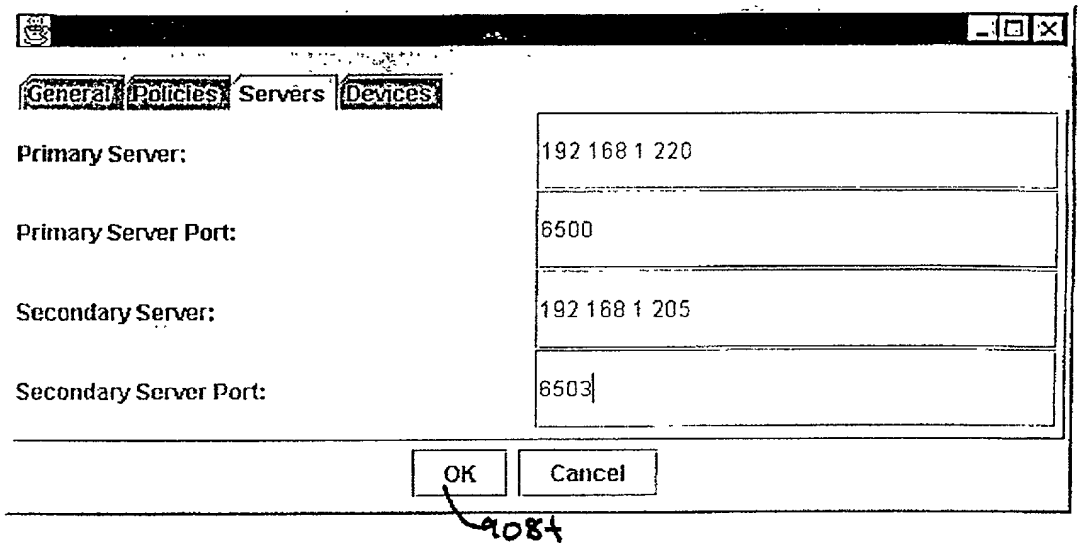

907
General
Username:
Kevin
908a
Description:
Kevin Snow user account
908e
Group Name:
Equipe
908f
Group Level Access:
908d
Password:

908b
Confirm Password:

908c
Policies
908h
User Cannot Change Password
908i
Account Disabled
908j
User Can Add Devices
908k
User Session Timeout:
15
Minutes
Servers
Primary Server:
192.168.1.220
908l
Primary Server Port:
6500
908n
Secondary Server:
192.168.1.221
908m
Secondary Server Port:
6503
908o
Devices
908p
908q
908r
908s
Device
READ
READ/WRITE
Retry
Timeou
Add
Delete
192.168.9.202
public
equipe
3
5
192.168.9.205
public
equipe
3
5
192.168.9.216
public
equipe
3
5
908g
OK
Cancel
908t

Fig. 11b

Fig. 11c



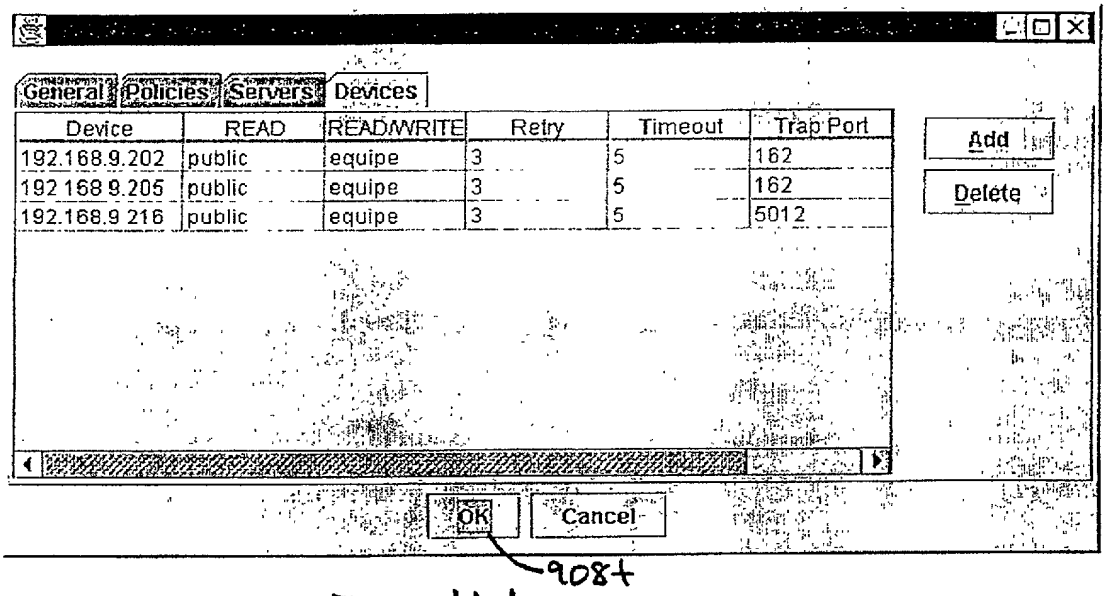
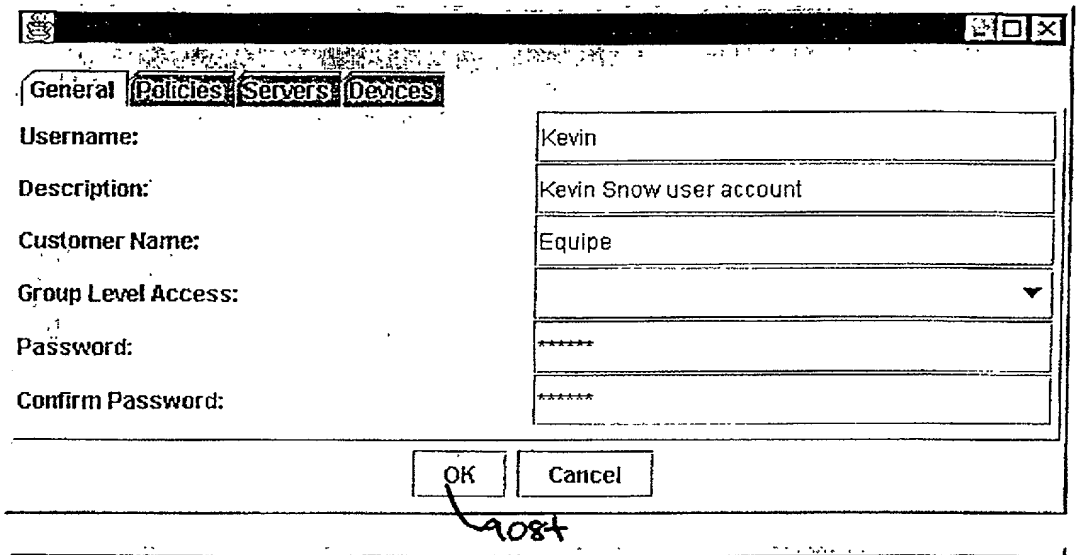


Fig. 11d

Fig. 11e



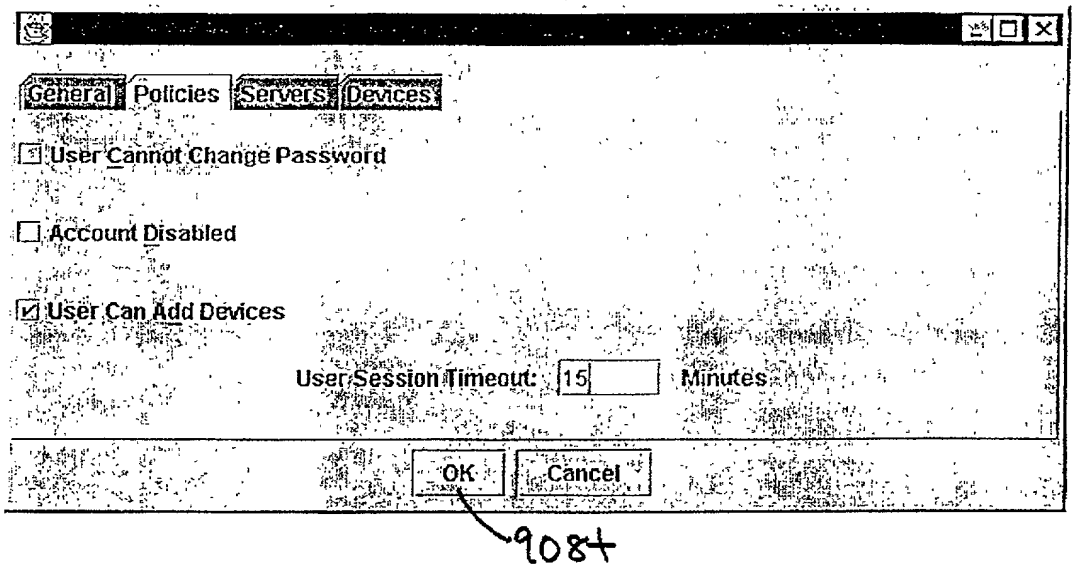


Fig. 11f

09/24/2000 15:09:04 #3

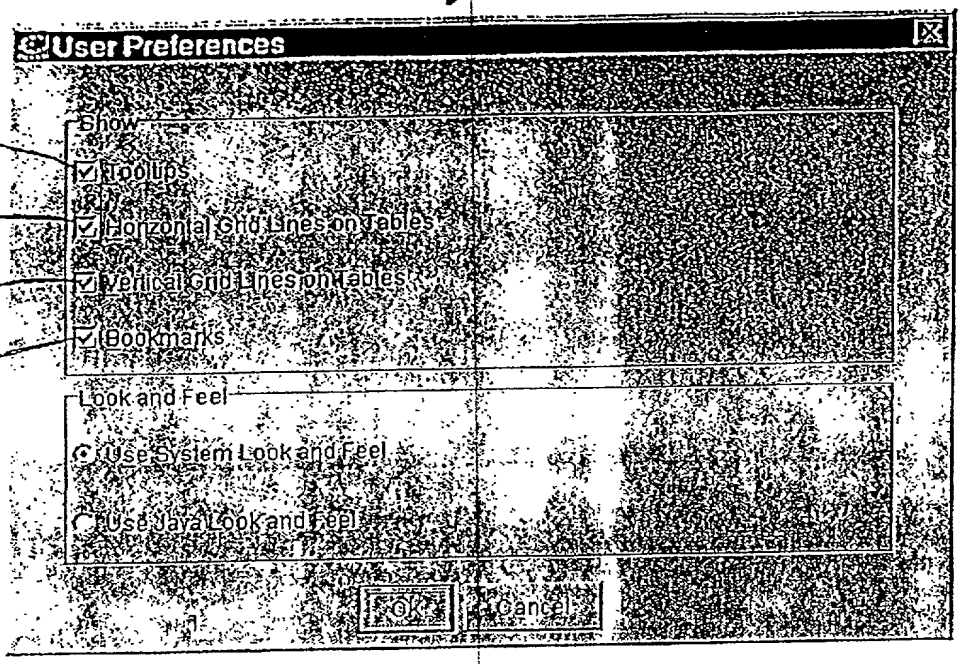
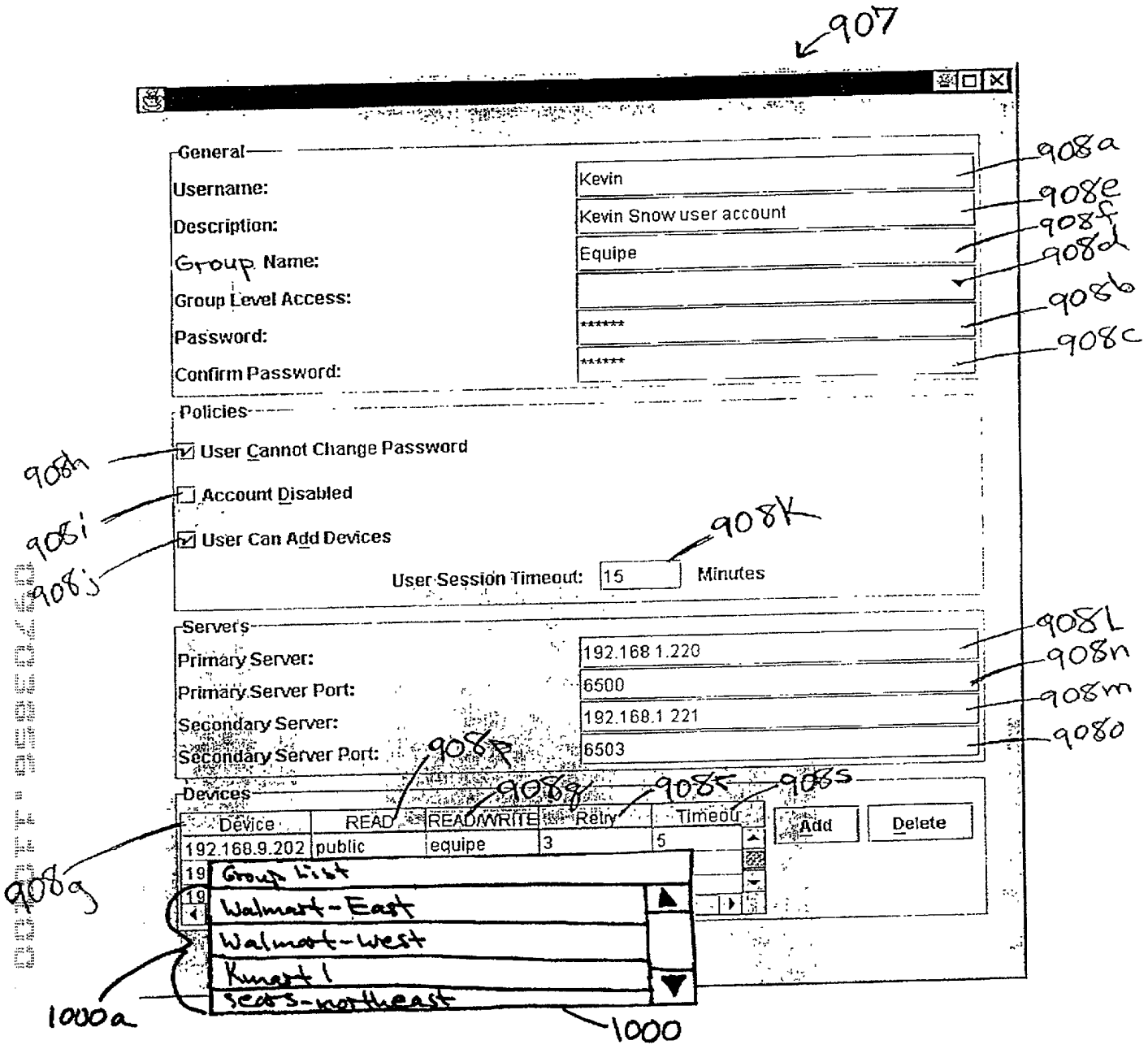

909
User Preferences
Show
910a
ToolTips
910b
Horizontal Grid Lines on Tables
910c
Vertical Grid Lines on Tables
910d
Bookmarks
Look and Feel
Use System Look and Feel
Use Java Look and Feel
OK
Cancel

Fig. 11g

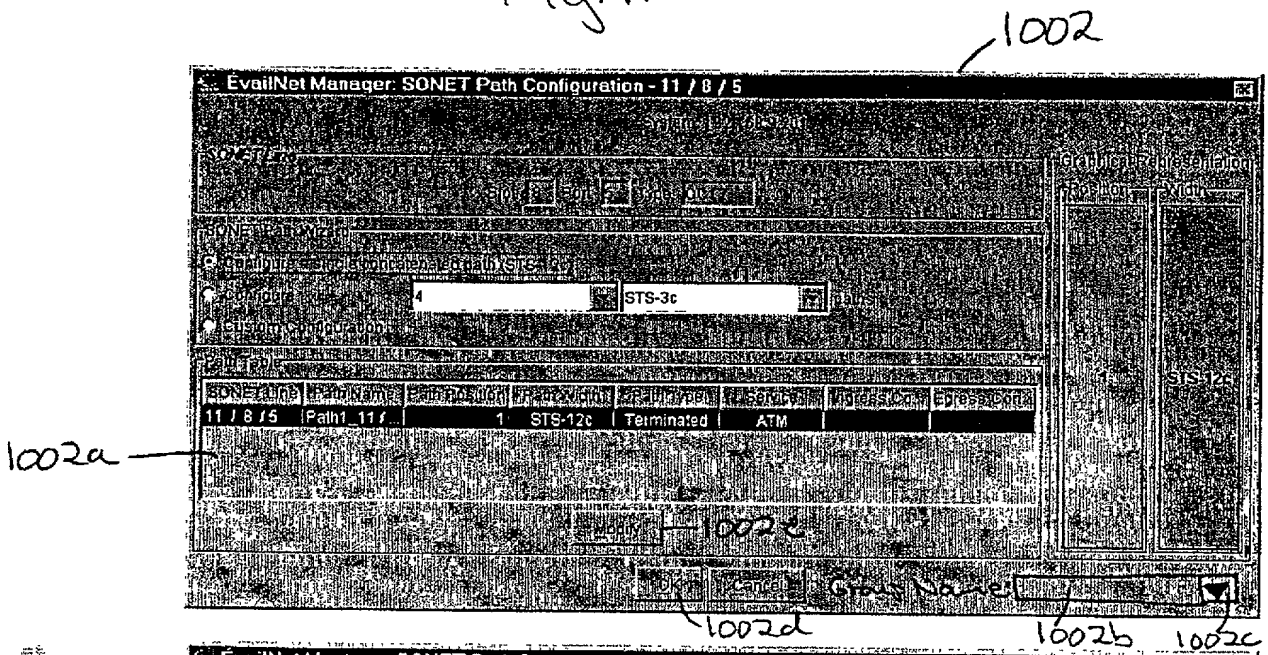

907
General
Username:
Kevin
908a
Description:
Kevin Snow user account
908e
Group Name:
Equipe
908f
Group Level Access:
908d
Password:

908b
Confirm Password:

908c
Policies
User Cannot Change Password
908h
Account Disabled
908i
User Can Add Devices
908j
User Session Timeout:
15
Minutes
908k
Servers
Primary Server:
192.168.1.220
908l
Primary Server Port:
6500
908n
Secondary Server:
192.168.1.221
908m
Secondary Server Port:
6503
908o
Devices
Device
READ
READ/WRITE
Retry
Timeout
908p
908q
908r
908s
Add
Delete
192.168.9.202
public
equipe
3
5
908g
Group List
Walmart-East
Walmart-west
Kmart1
sears-northeast
1000a
1000

Fig. 11h

Fig. 11i



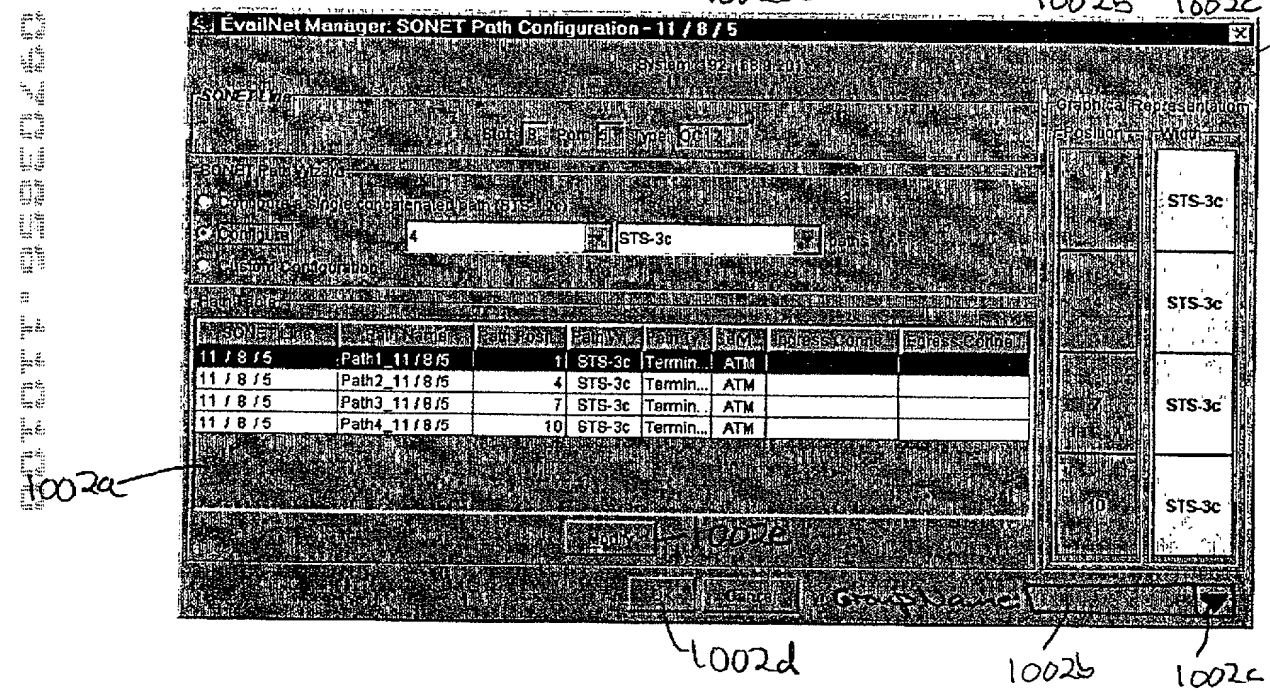
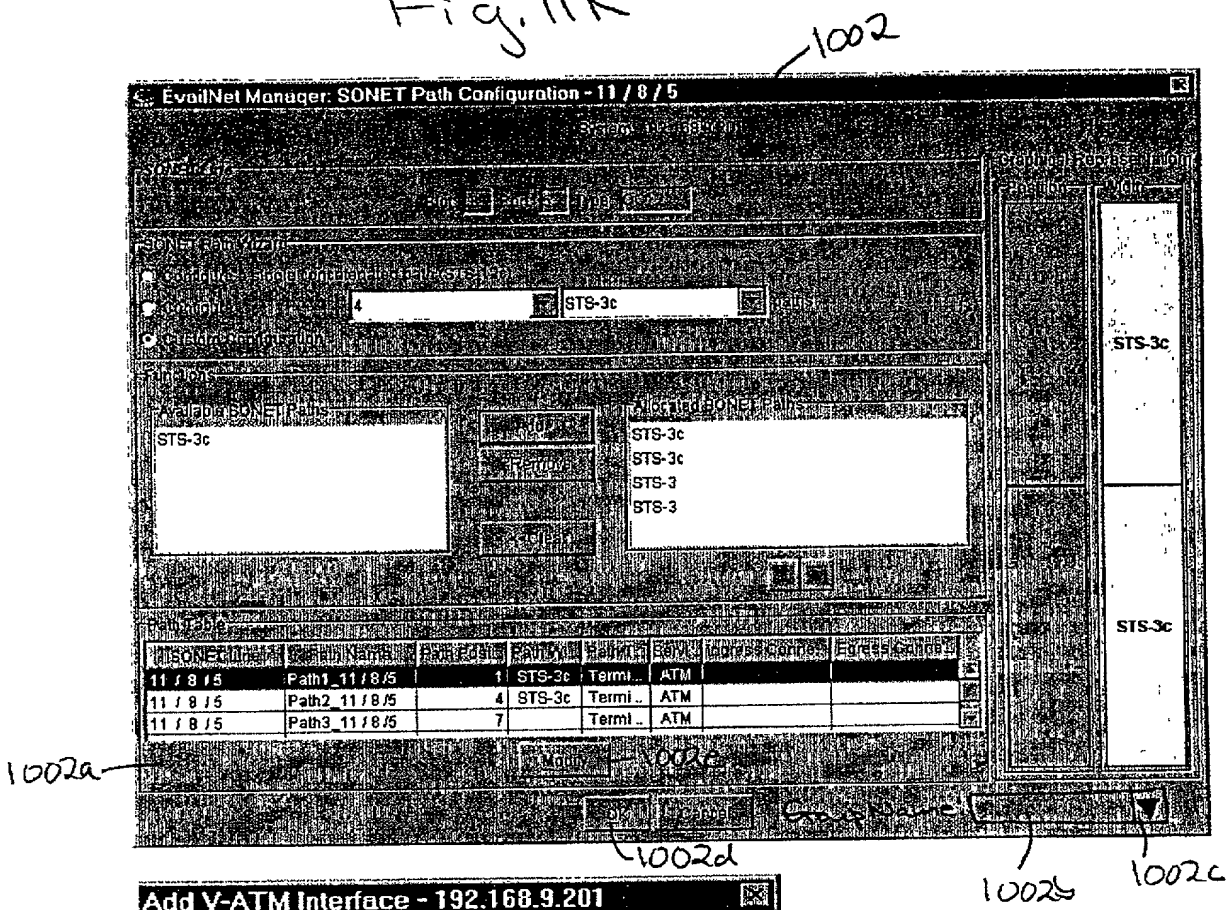


Fig. 11j

Fig. 11K



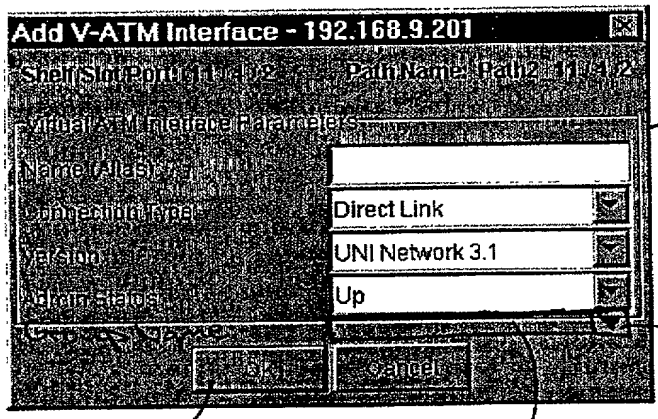


Fig. 11n

Fig. 11L

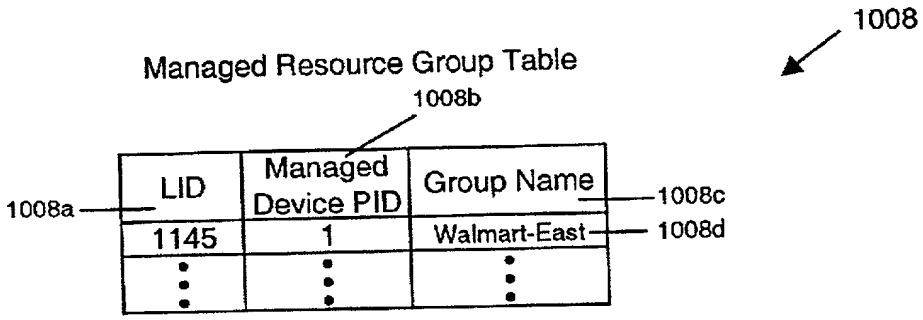
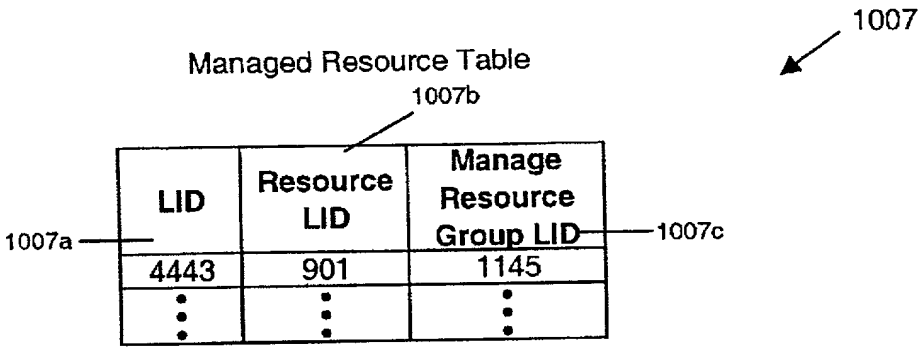


Fig. 11M



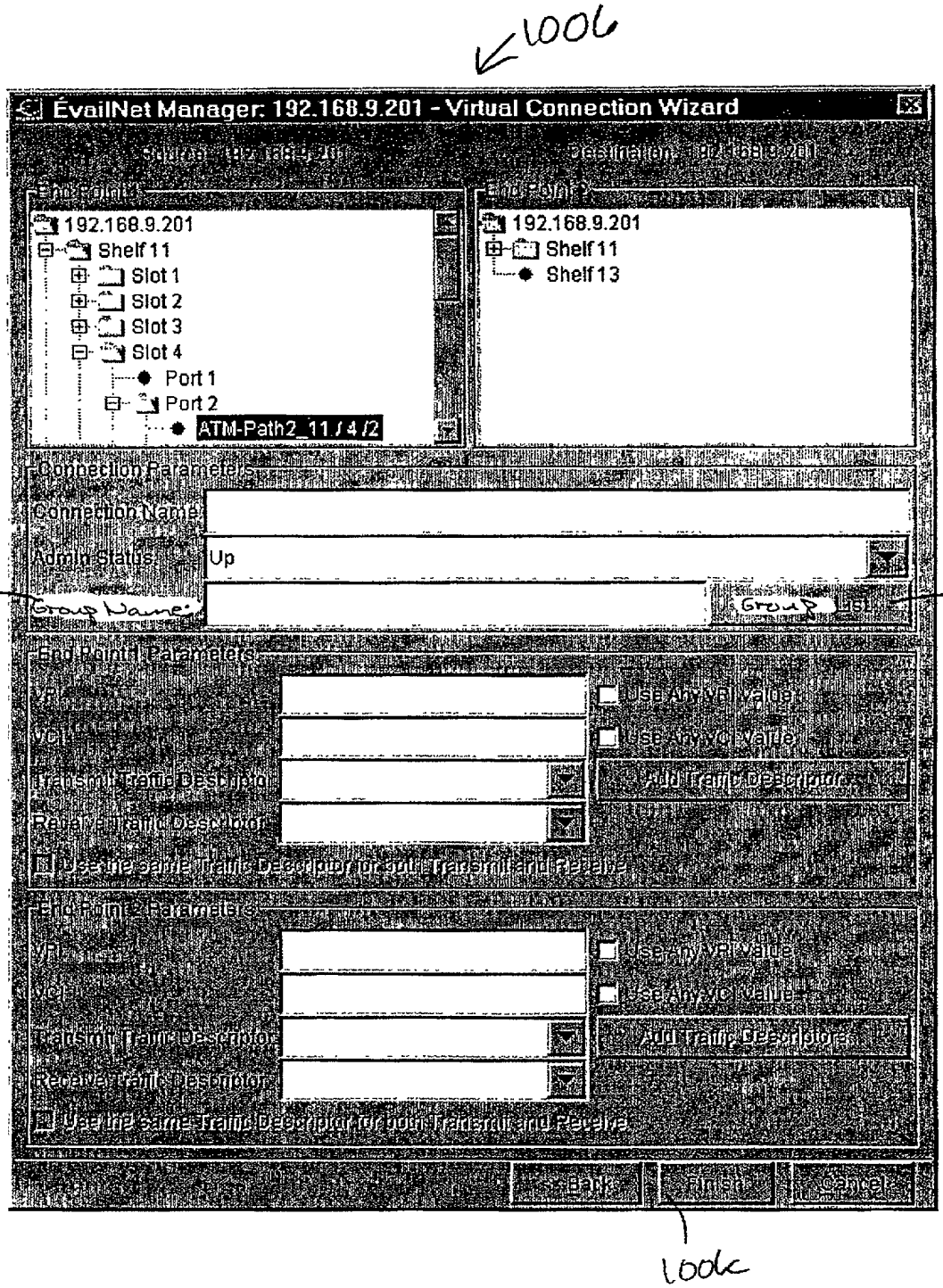

1006
EvailNet Manager: 192.168.9.201 - Virtual Connection Wizard
End Point 1
192.168.9.201
Shelf 11
Slot 1
Slot 2
Slot 3
Slot 4
Port 1
Port 2
ATM-Path2_11 / 4 /2
End Point 2
192.168.9.201
Shelf 11
Shelf 13
Connection Parameters
Connection Name:
Admin Status:
Up
Group Name:
Group List
1006a
1006b
End Point1 Parameters
Use Any VPI value
Use Any VCI value
Transmit Traffic Descriptor:
Receive Traffic Descriptor:
Add Traffic Descriptor
Use the same Traffic Descriptor for both Transmit and Receive
End Point 2 Parameters
Use Any VPI value
Use Any VCI value
Transmit Traffic Descriptor:
Receive Traffic Descriptor:
Add Traffic Descriptor
Use the same Traffic Descriptor for both Transmit and Receive
< Back
Finish
Cancel
1006c

Fig. 11a

Fig. 11p

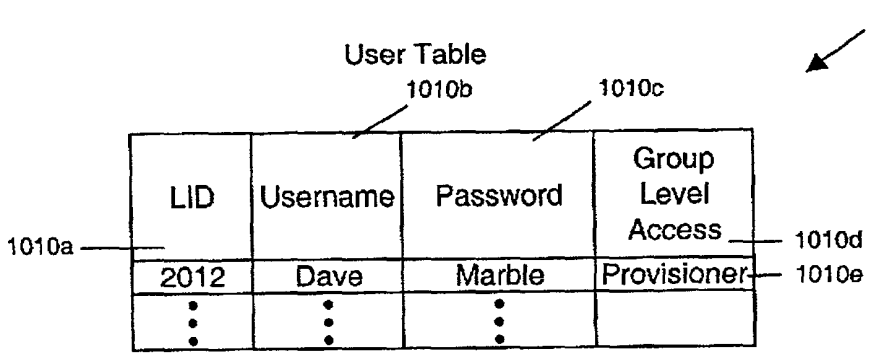


Fig. 11q

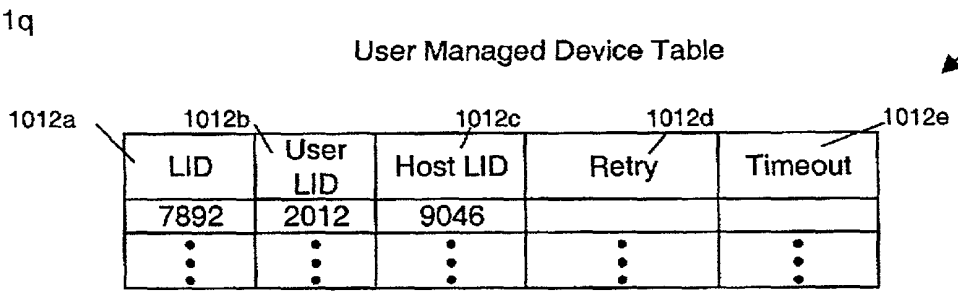


Fig. 11r

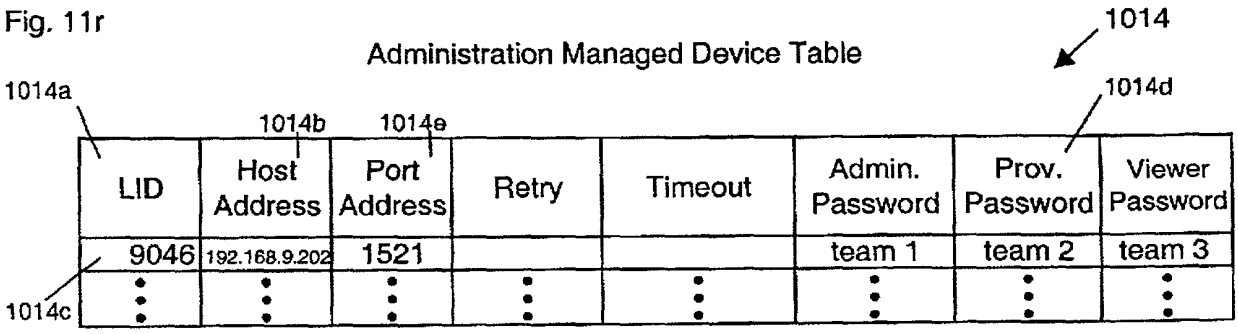


Fig. 11s

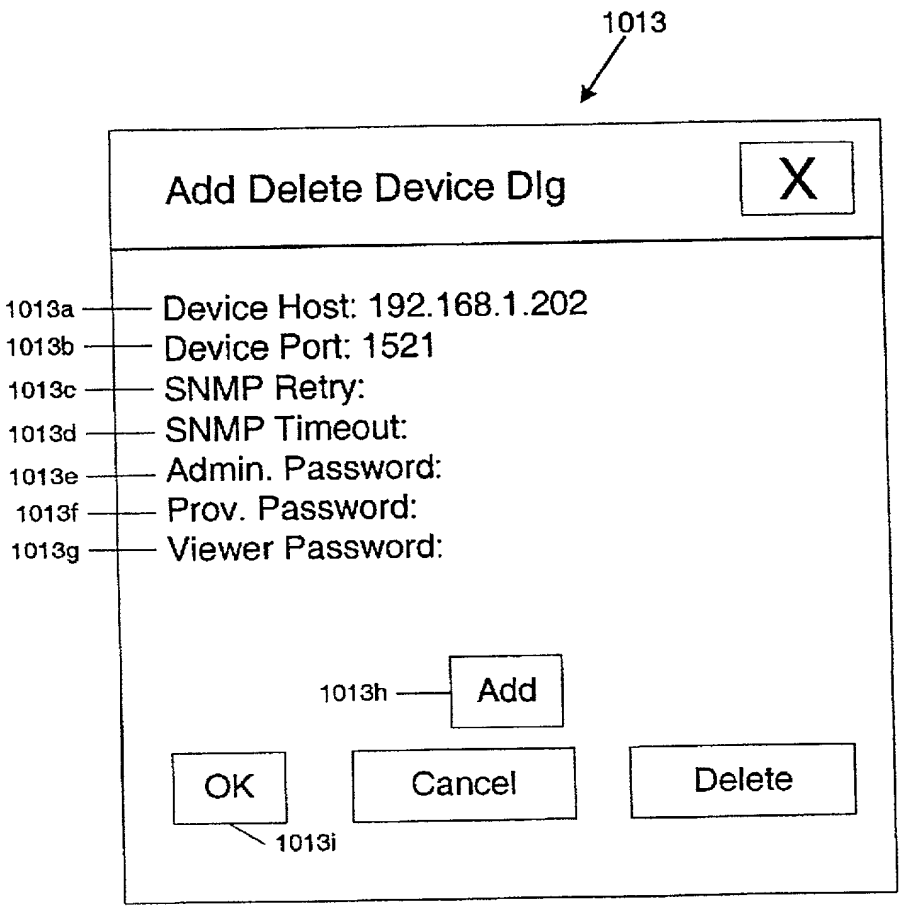

1013
Add Delete Device Dlg
X
1013a
Device Host: 192.168.1.202
1013b
Device Port: 1521
1013c
SNMP Retry:
1013d
SNMP Timeout:
1013e
Admin. Password:
1013f
Prov. Password:
1013g
Viewer Password:
1013h
Add
OK
Cancel
Delete
1013i

Fig. 11t

User Resource Group Map Table

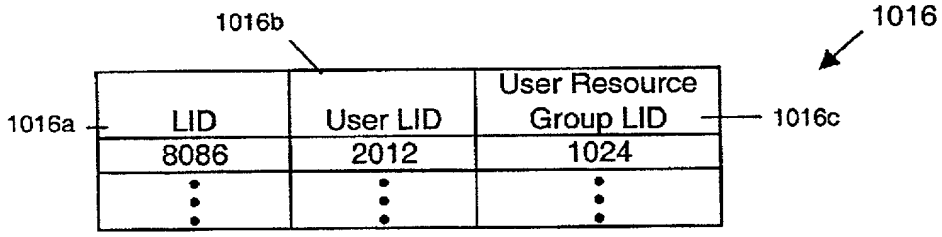
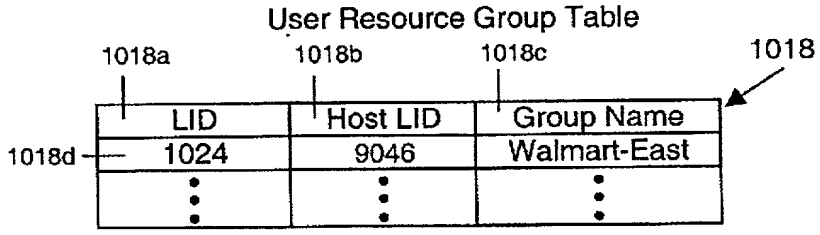


Fig. 11u

User Resource Group Table



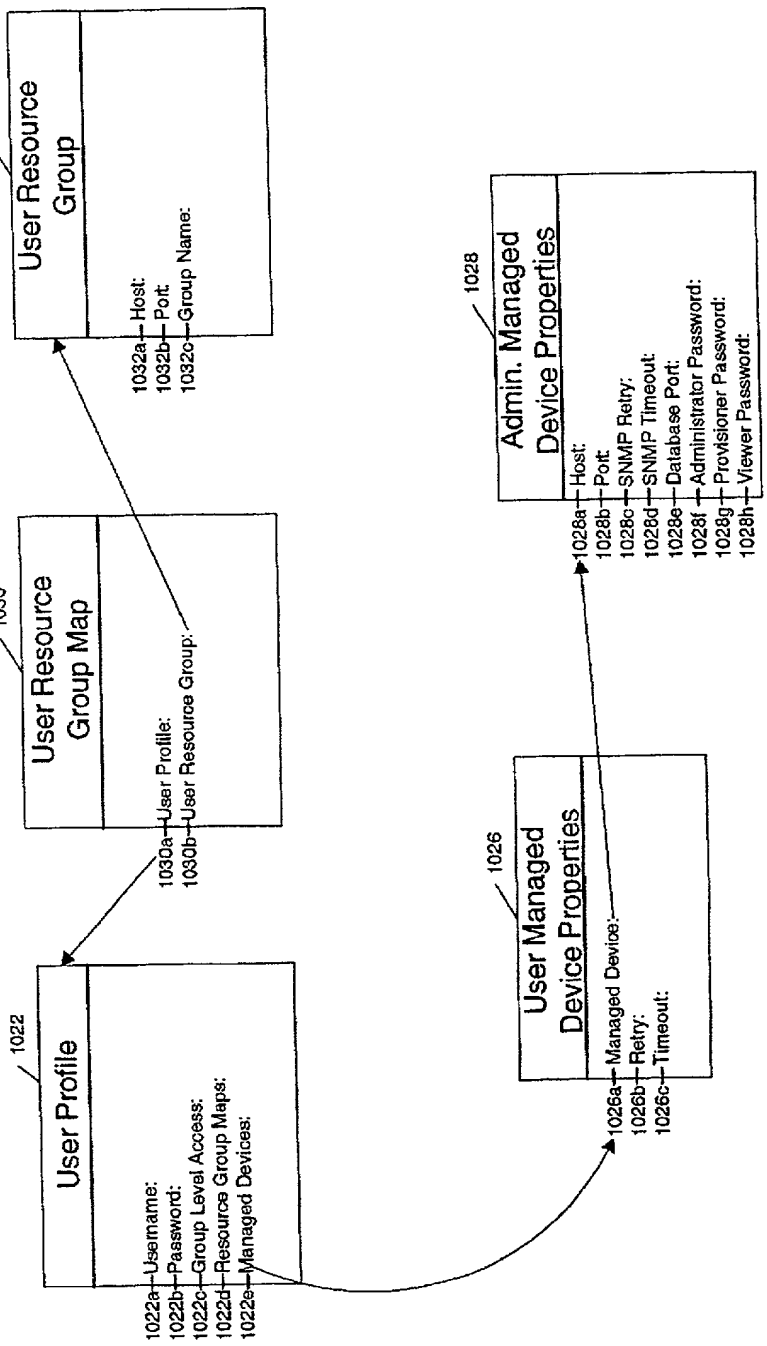

User Profile
1022
1022a Username:
1022b Password:
1022c Group Level Access:
1022d Resource Group Maps:
1022e Managed Devices:
User Resource Group Map
1030
1030a User Profile:
1030b User Resource Group:
User Resource Group
1032
1032a Host:
1032b Port:
1032c Group Name:
User Managed Device Properties
1026
1026a Managed Device:
1026b Retry:
1026c Timeout:
Admin. Managed Device Properties
1028
1028a Host:
1028b Port:
1028c SNMP Retry:
1028d SNMP Timeout:
1028e Database Port:
1028f Administrator Password:
1028g Provisioner Password:
1028h Viewer Password:

Fig. 11v

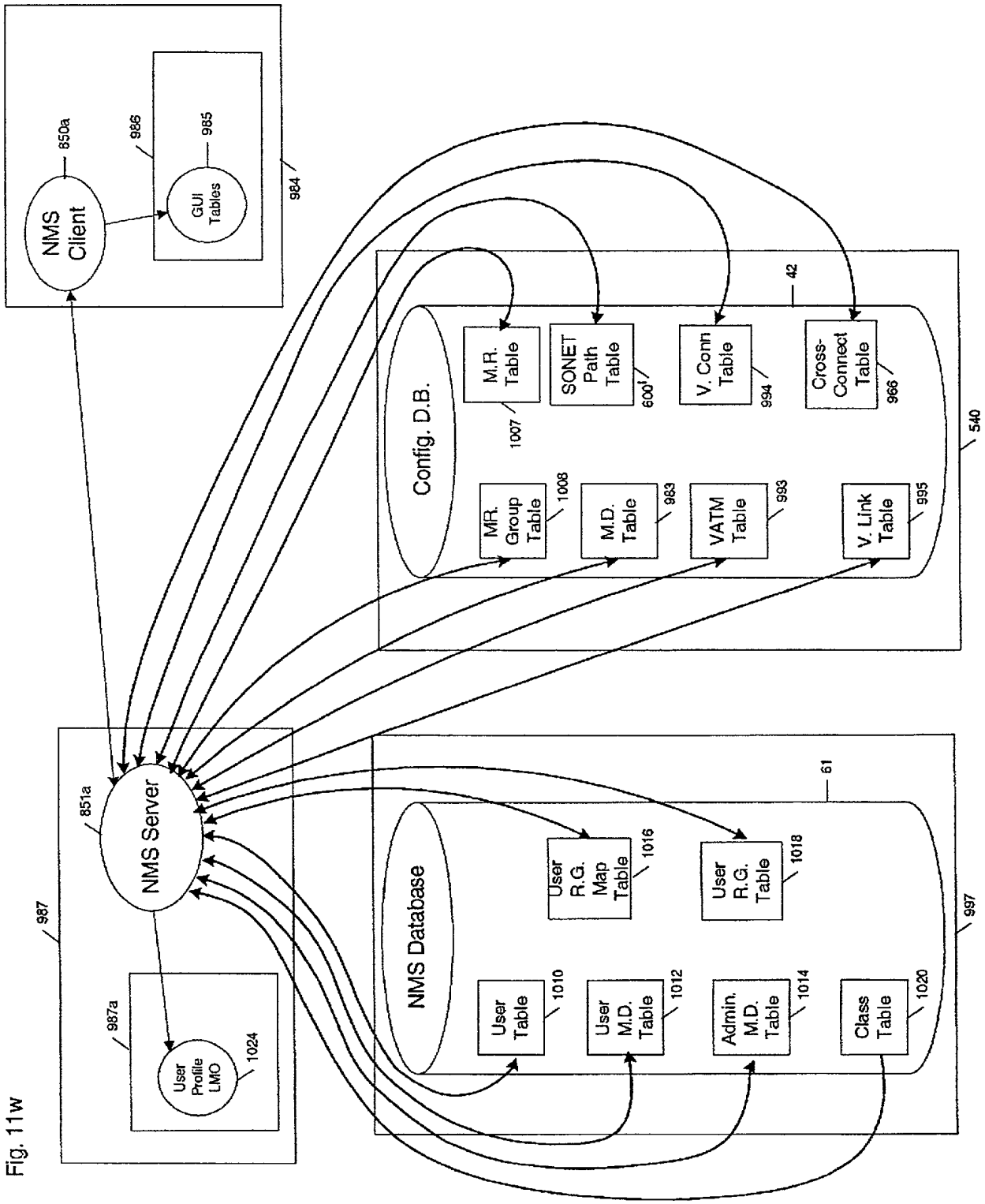
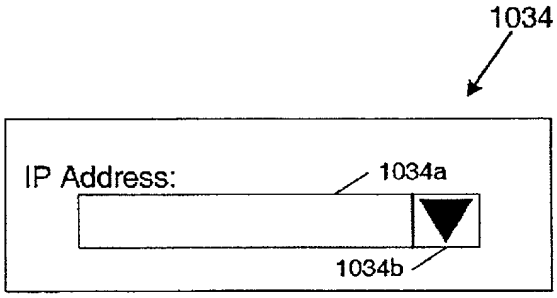

NMS Client
850a
GUI Tables
985
986
984
Config. D.B.
M.R. Table
1007
SONET Path Table
600
V. Conn Table
994
Cross-Connect Table
966
42
MR. Group Table
1008
M.D. Table
983
VATM Table
993
V. Link Table
995
540
NMS Server
851a
987
User Profile LMO
1024
987a
NMS Database
User R.G. Map Table
1016
User R.G. Table
1018
User Table
1010
User M.D. Table
1012
Admin. M.D. Table
1014
Class Table
1020
61
997

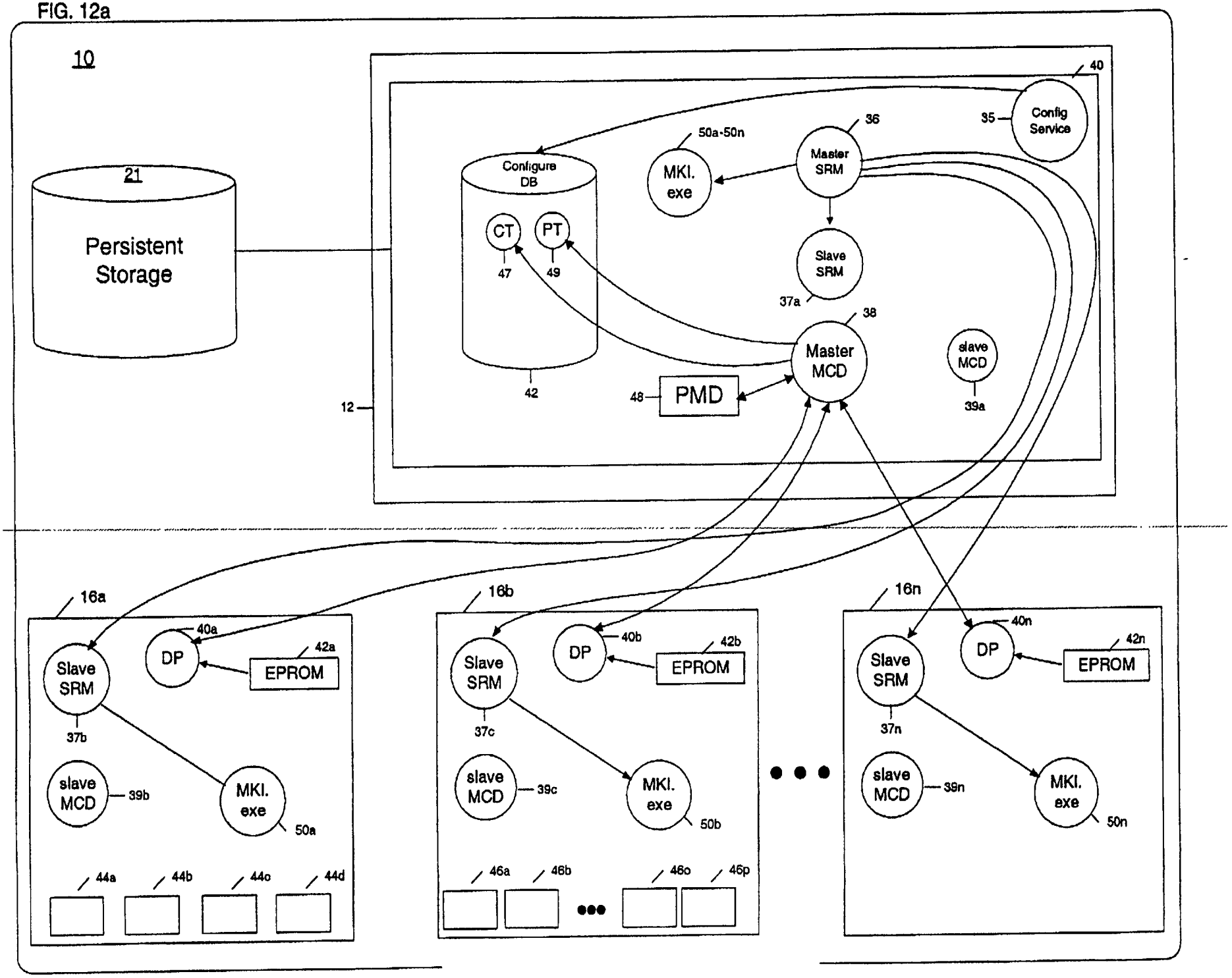
Fig. 11w

Fig. 11x



SCANNED. #_6

FIG. 12a



Sent By: Equipe Communications; 781 795 2031; Sep-15-00 2:48PM; Page 14/21

SCANNED # 6

Sent By: Equipe Communications; 781 795 2031; Sep-15-00 2:48PM; Page 15/21

CARD TABLE

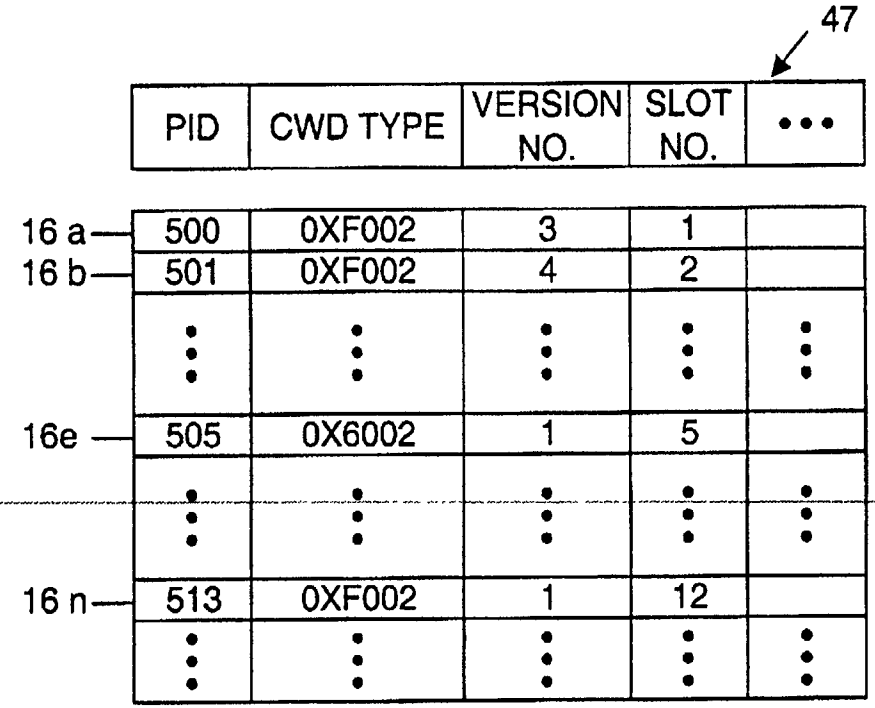


FIG. 12b

SCANNED. # 6

Page 16/21
Sep-15-00 2:48PM;
781 795 2031;
Sent By: Equipe Communications;

PORT TABLE

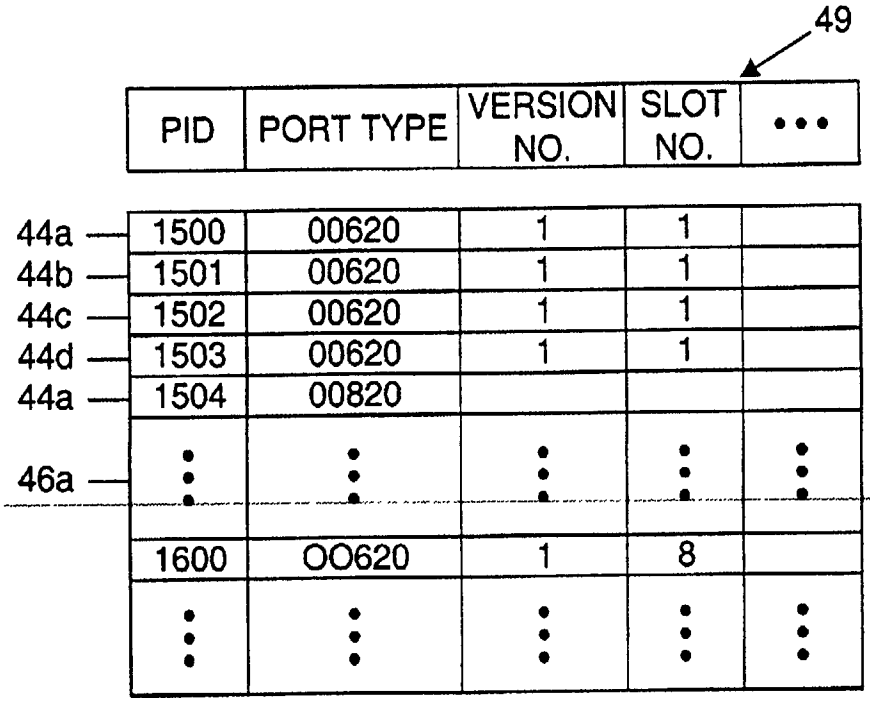
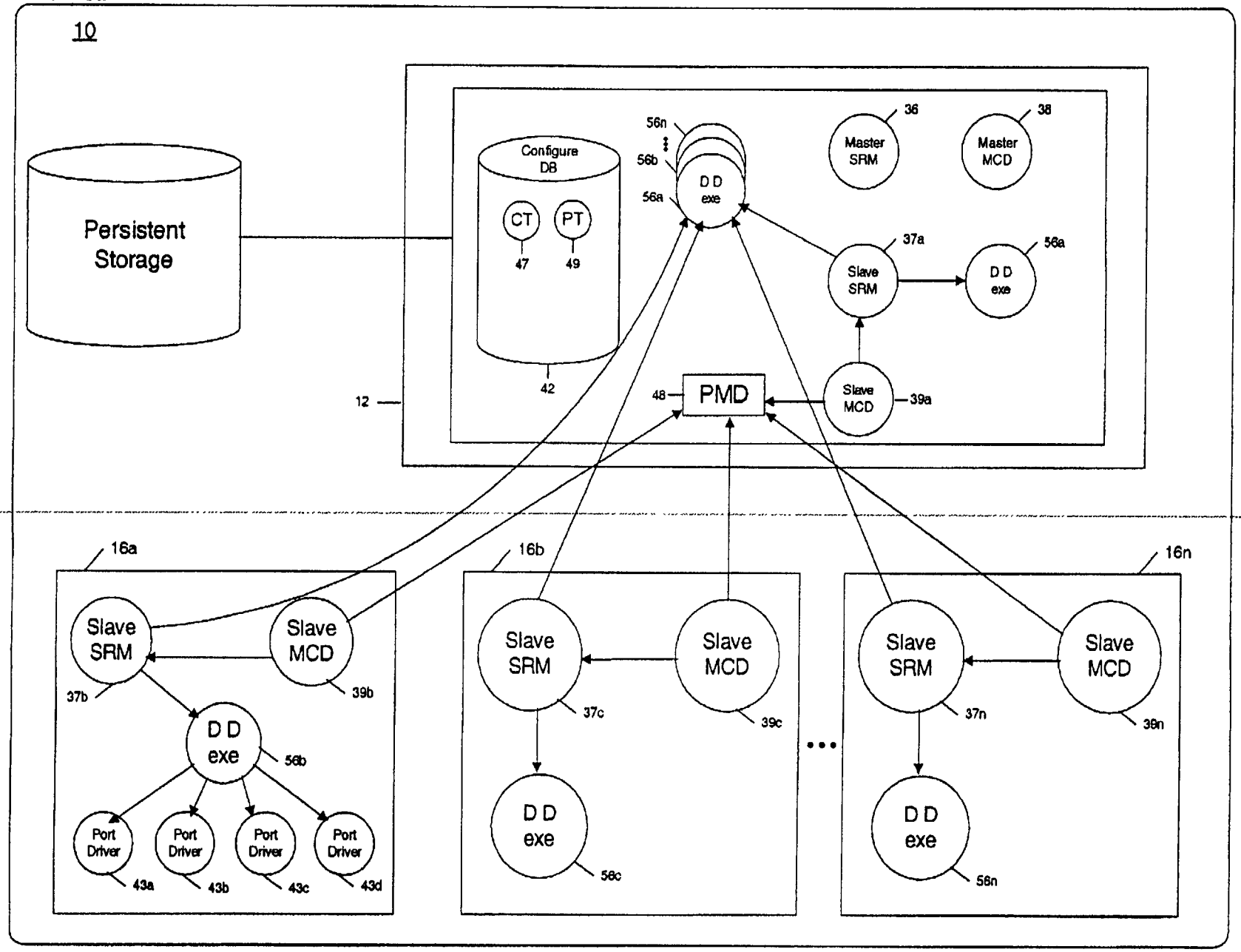


FIG. 12c

SCANNED. # 6

Sent By: Equipe Communications; 781 795 2031; Sep-15-00 2:48PM; Page 17/21

FIG. 13a



SCANNED # 6

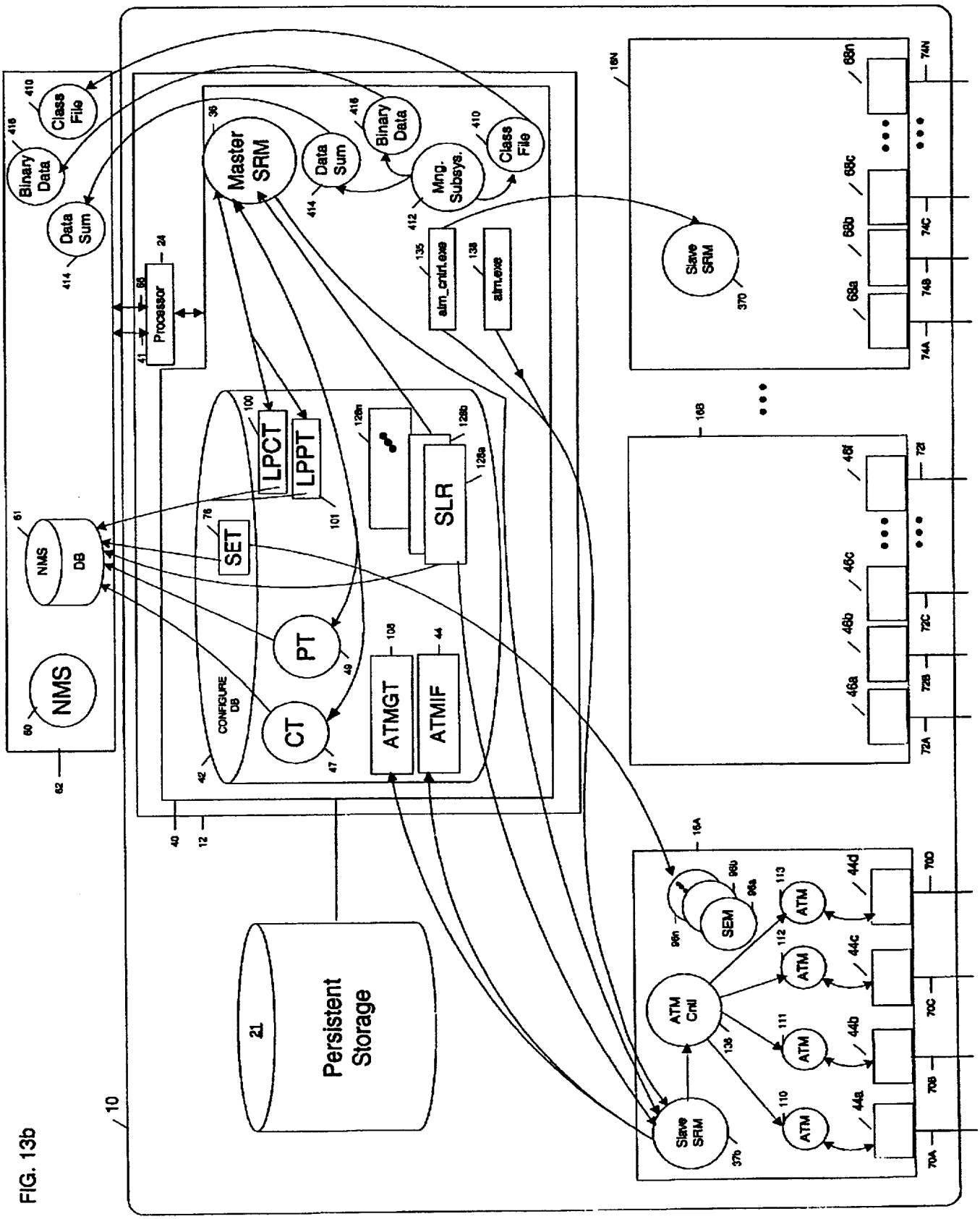


FIG. 13b

SCANNED, # 6

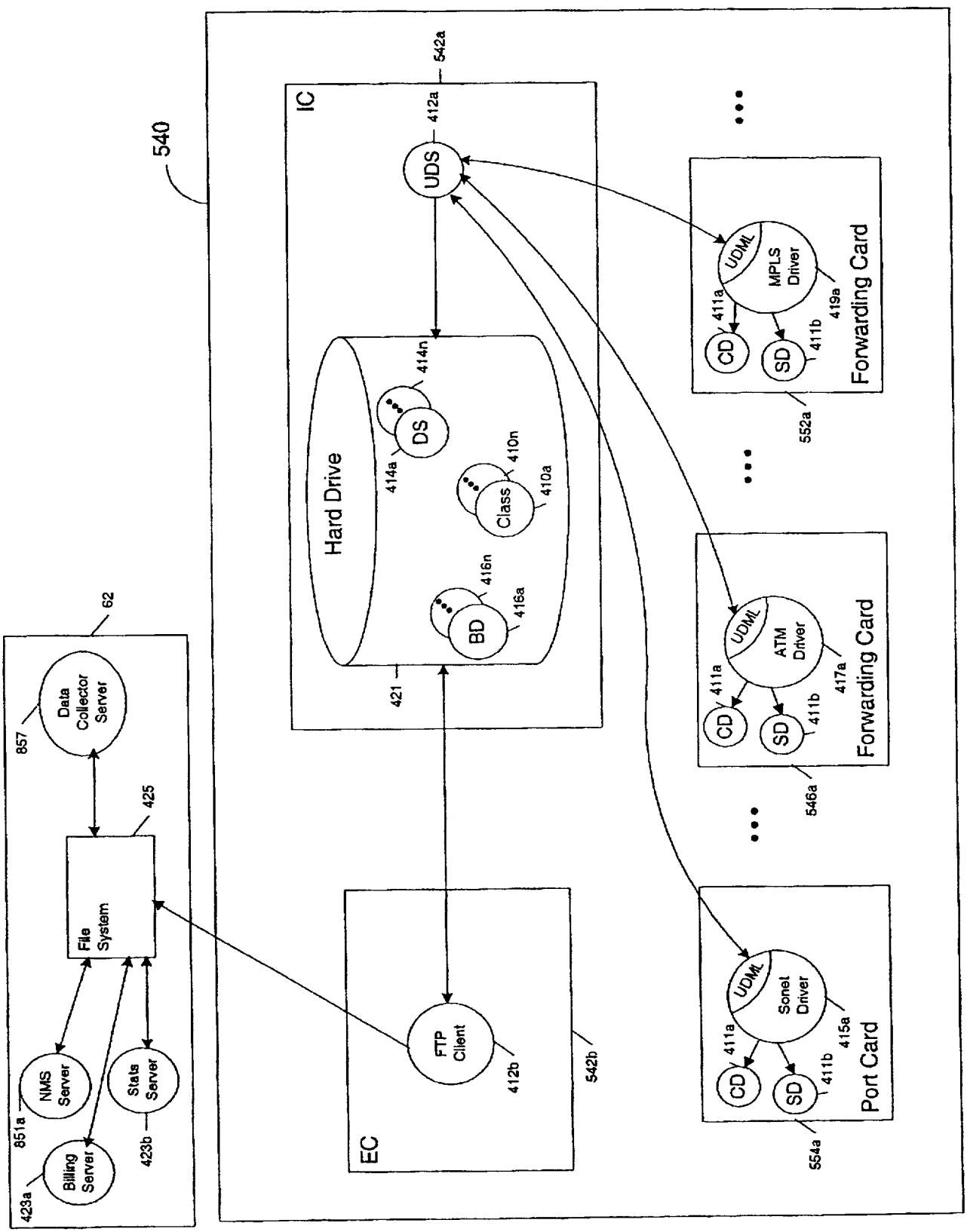


Fig. 13c

SCANNED, # 6

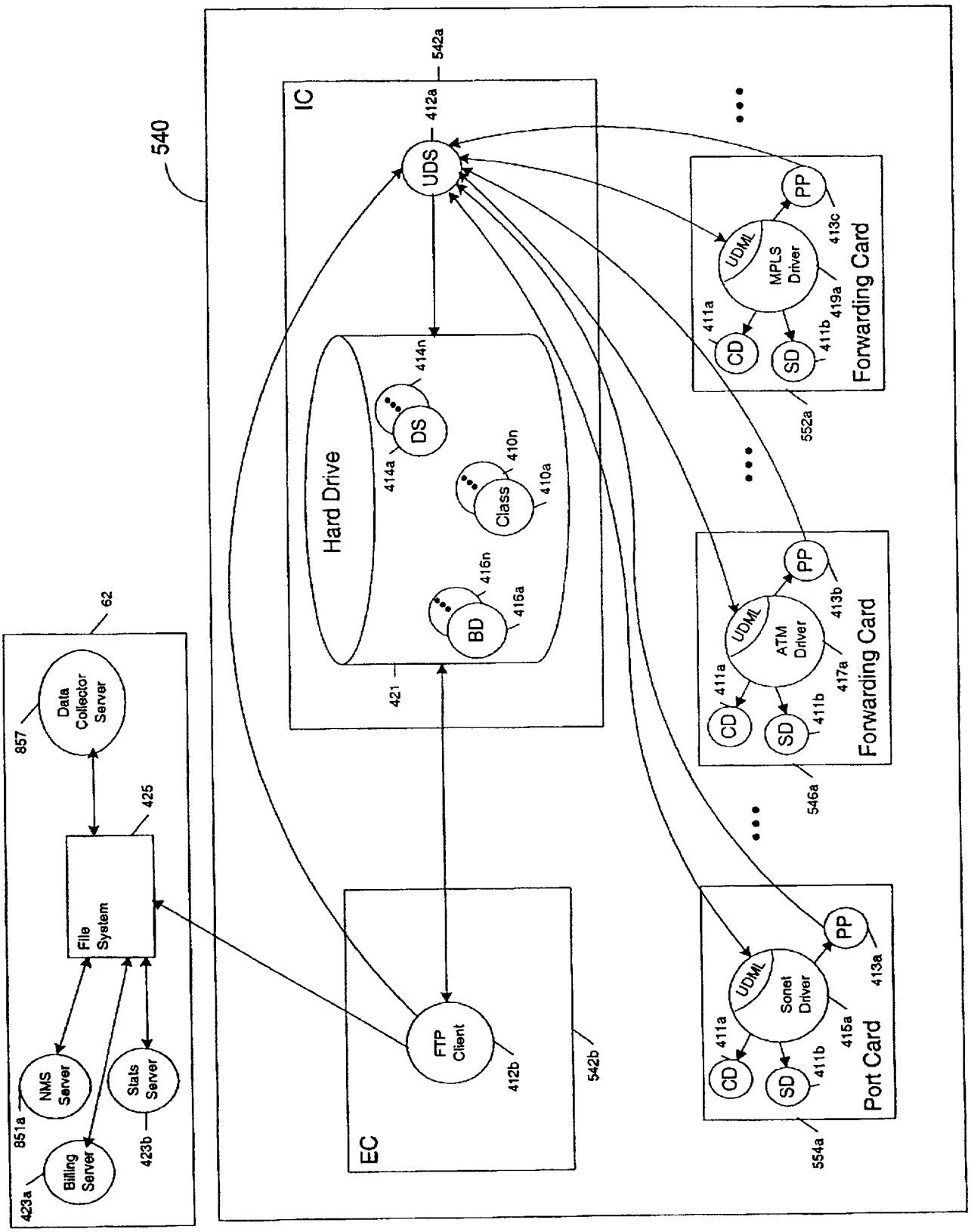


Fig. 13d

FIG. 14a

Service Endpoint Table 76

	Service Endpoint #	Port PID
78	1	1500
80	2	1501
82	3	1501
84	4	1501
86	5	1502
88	6	1502
90	7	1503
92	8	1503
94	9	1503
168	10	1502
	⋮	⋮

FIG. 14b

Logical to Physical Card Table 100

	LID (98)	Primary PID (102)	Back-up PID (104)
106	30	500	513
109	31	501	513
	⋮	⋮	⋮

FIG. 14c

Logical to Physical Port Table 101

	LID (98)	Primary PID (102)	Back-up PID (104)
107	40	1500	1600
	⋮	⋮	⋮

FIG. 14d

ATM Group Table 108

Group #	Card LID	•••
1	30	
2	30	
3	30	
4	30	

FIG. 14e

ATM Interface Table 114

ATM IF	ATM Group	SE	•••
1	1	1	
2	1	1	
3	1	1	
4	2	2	
5	2	3	
6	2	4	
⋮	⋮	⋮	⋮
12	3	10	
⋮	⋮	⋮	⋮

170

FIG. 14f

Software Load Record 128a

130 Control Shim	LID 132
134 atm-cntl.exe	30

FIG. 15

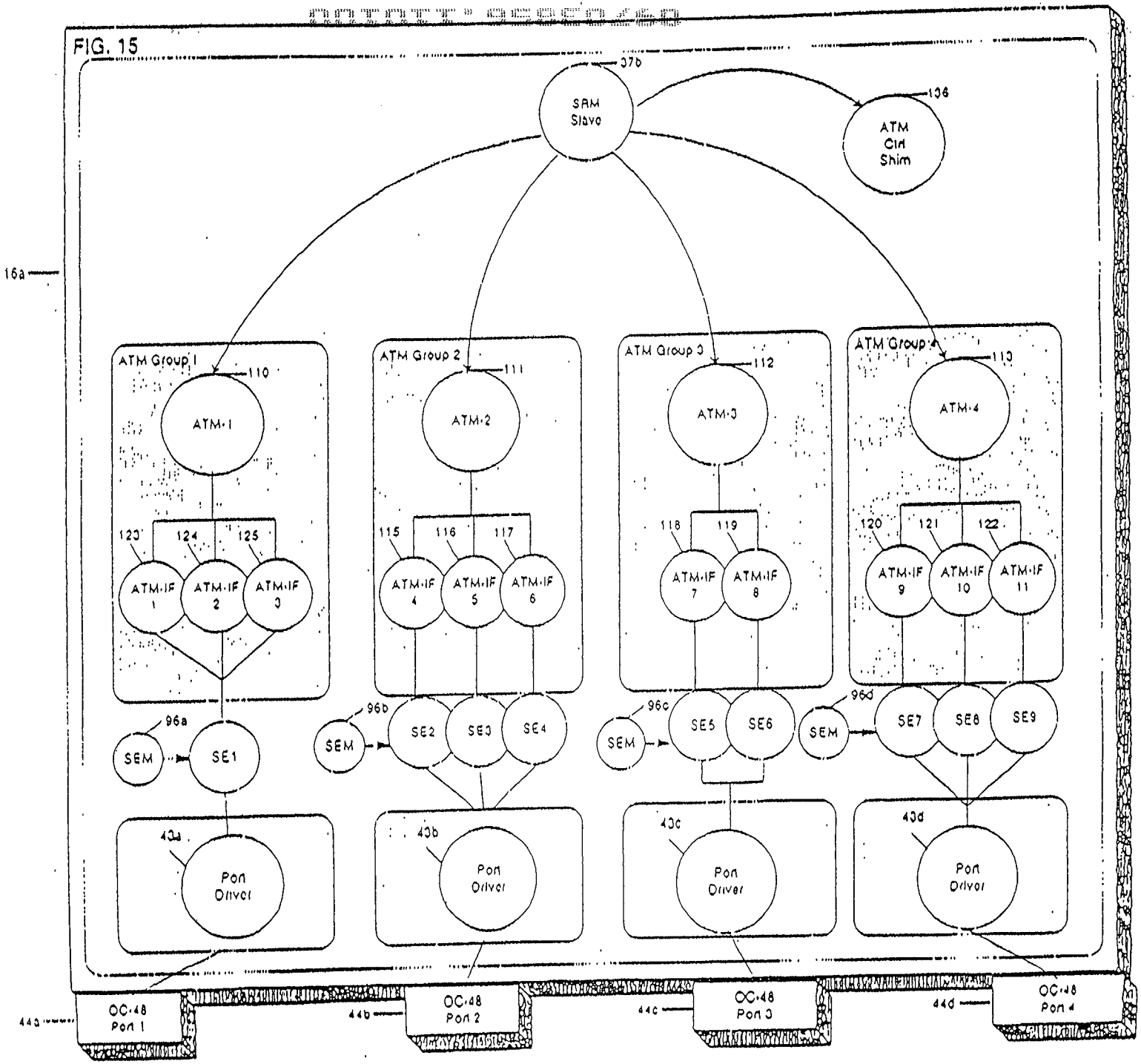

37b
SAM Slave
106
ATM Clr Shim
16a
ATM Group 1
110
ATM-1
120
124
125
ATM-IF 1
ATM-IF 2
ATM-IF 3
96a
SEM
SE1
43a
Pon Driver
ATM Group 2
111
ATM-2
115
116
117
ATM-IF 4
ATM-IF 5
ATM-IF 6
96b
SEM
SE2
SE3
SE4
43b
Pon Driver
ATM Group 3
112
ATM-3
118
119
ATM-IF 7
ATM-IF 8
96c
SEM
SE5
SE6
43c
Pon Driver
ATM Group 4
113
ATM-4
120
121
122
ATM-IF 9
ATM-IF 10
ATM-IF 11
96d
SEM
SE7
SE8
SE9
43d
Pon Driver
44a
OC-48 Pon 1
44b
OC-48 Pon 2
44c
OC-48 Pon 3
44d
OC-48 Pon 4

FIG. 16a

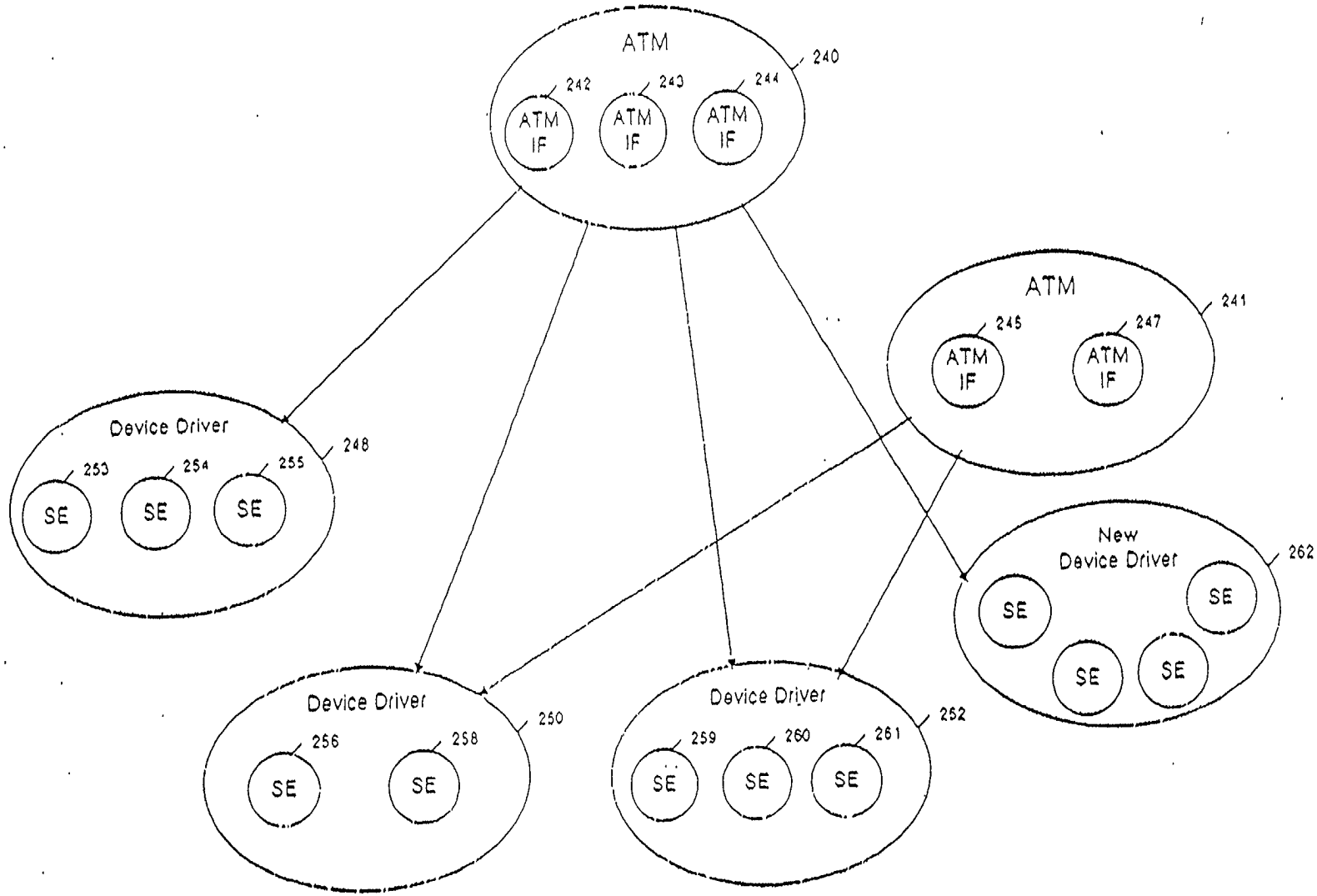
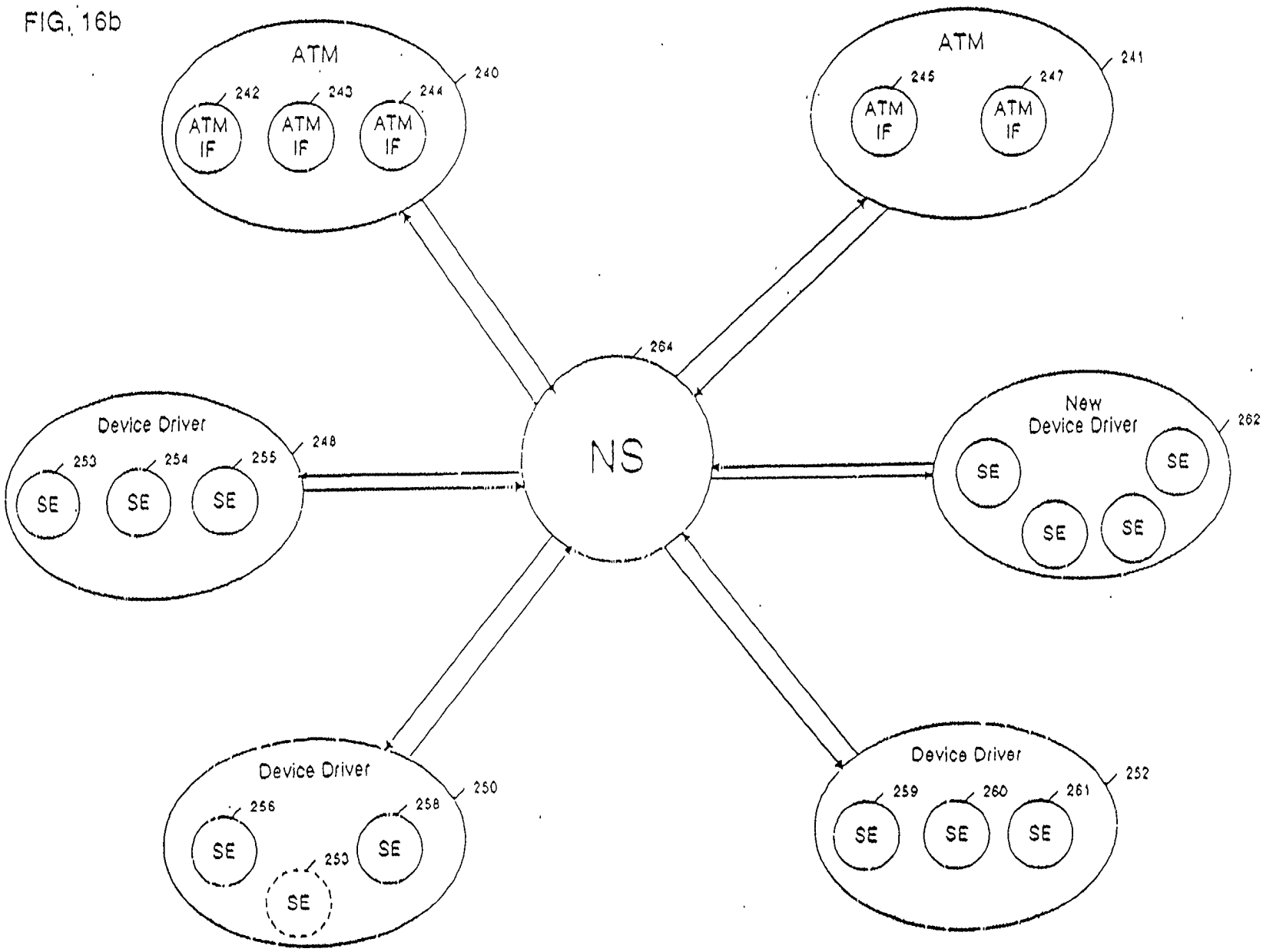

ATM
240
242
243
244
ATM IF
ATM IF
ATM IF
ATM
241
245
247
ATM IF
ATM IF
Device Driver
248
253
254
255
SE
SE
SE
New Device Driver
262
SE
SE
SE
SE
Device Driver
250
256
258
SE
SE
Device Driver
252
259
260
261
SE
SE
SE

FIG. 16b


ATM
240
242
ATM IF
243
ATM IF
244
ATM IF
ATM
241
245
ATM IF
247
ATM IF
264
NS
Device Driver
248
253
SE
254
SE
255
SE
New Device Driver
262
SE
SE
SE
SE
Device Driver
250
256
SE
258
SE
257
SE
Device Driver
252
259
SE
260
SE
261
SE

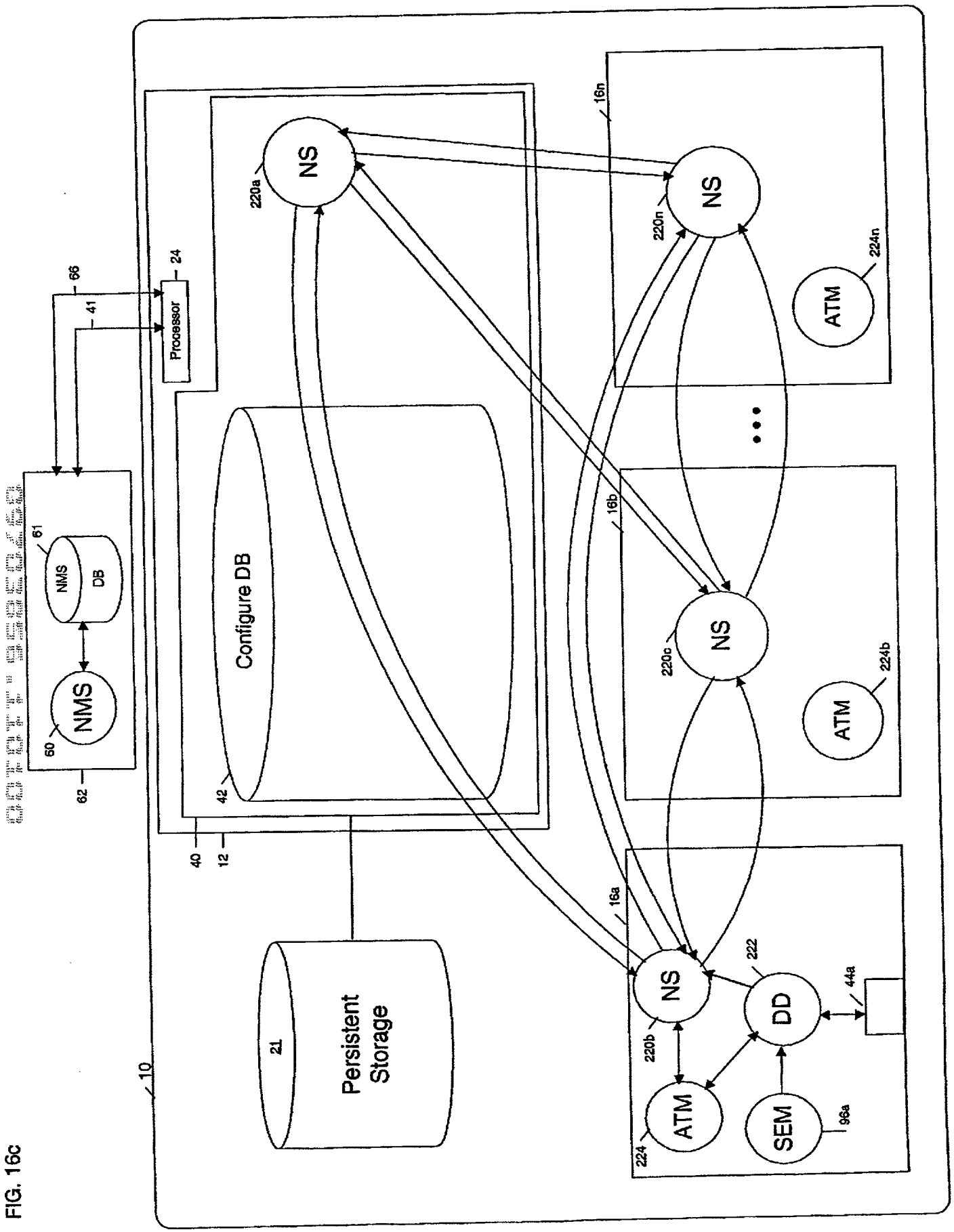
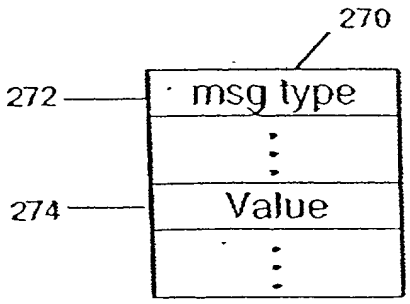

10
Persistent Storage
21
Configure DB
42
Processor
24
66
41
NMS
DB
61
NMS
60
62
40
12
NS
220a
NS
220n
ATM
224n
16n
NS
220c
ATM
224b
16b
NS
220b
DD
222
44a
ATM
224
SEM
96a
16a

FIG. 16c

FIG. 16d



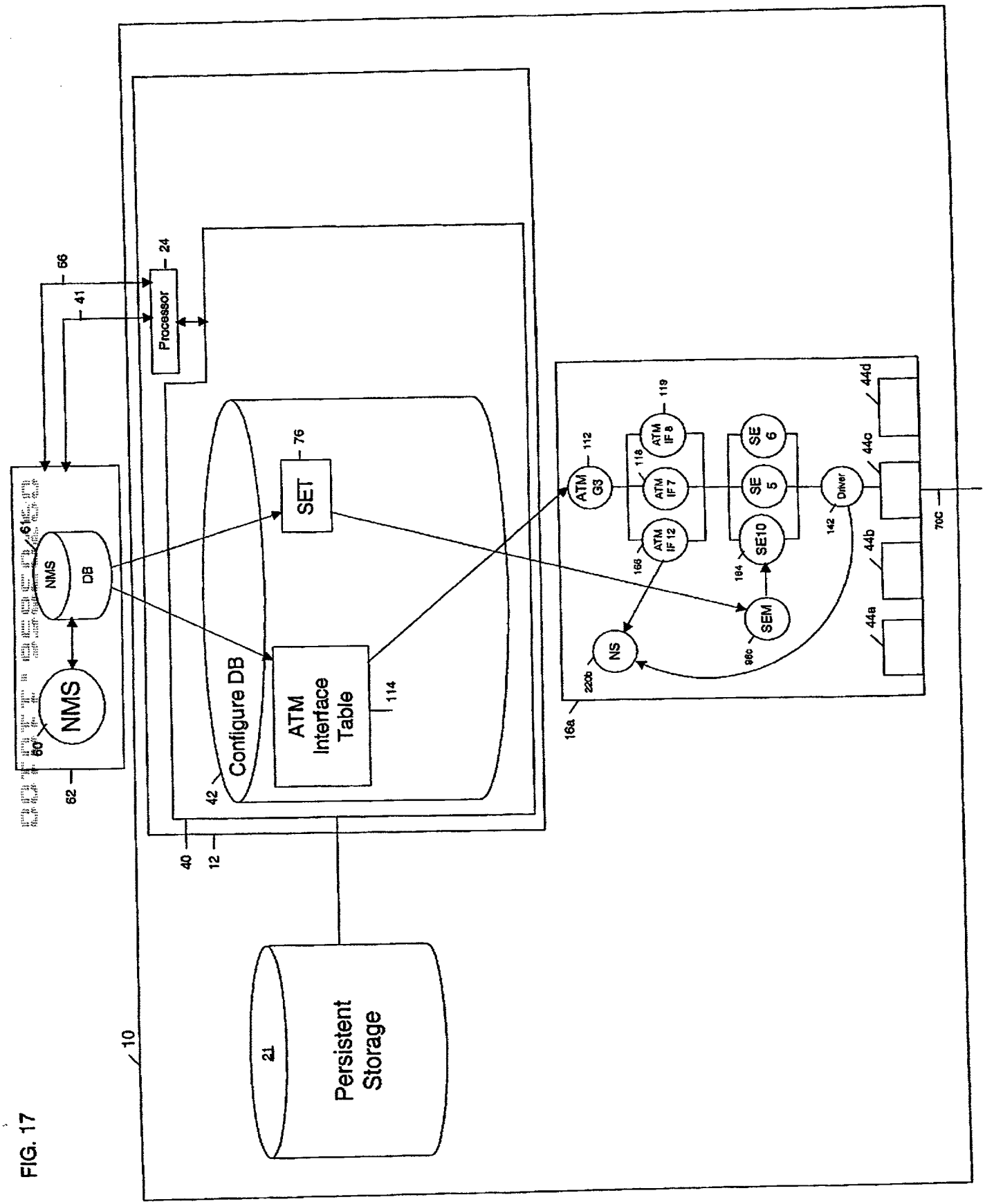

66
41
24
Processor
NMS
DB
61
NMS
60
62
SET
76
Configure DB
ATM Interface Table
114
42
40
12
Persistent Storage
21
10
ATM G3
112
ATM IF 8
119
ATM IF 7
118
ATM IF 12
166
SE 6
SE 5
SE10
184
SEM
96c
NS
220b
Driver
142
44d
44c
44b
44a
70C
16a

FIG. 17

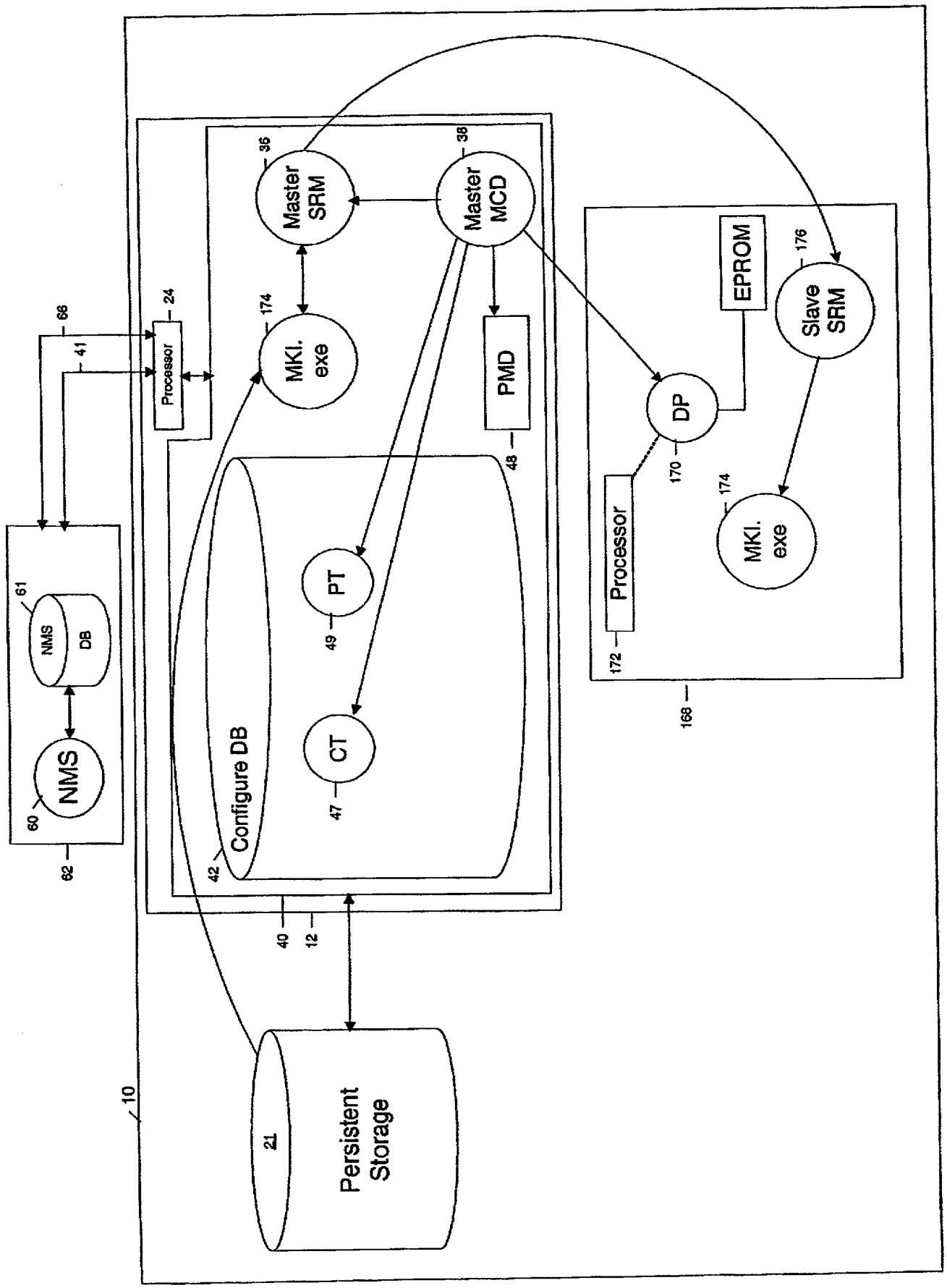

10
Persistent Storage
21
Master SRM
36
Master MCD
38
MKI. exe
174
PMD
48
Processor
24
66
41
PT
49
CT
47
Configure DB
42
40
12
NMS
DB
61
NMS
60
62
DP
170
EPROM
Slave SRM
176
Processor
172
MKI. exe
174
168

FIG. 18

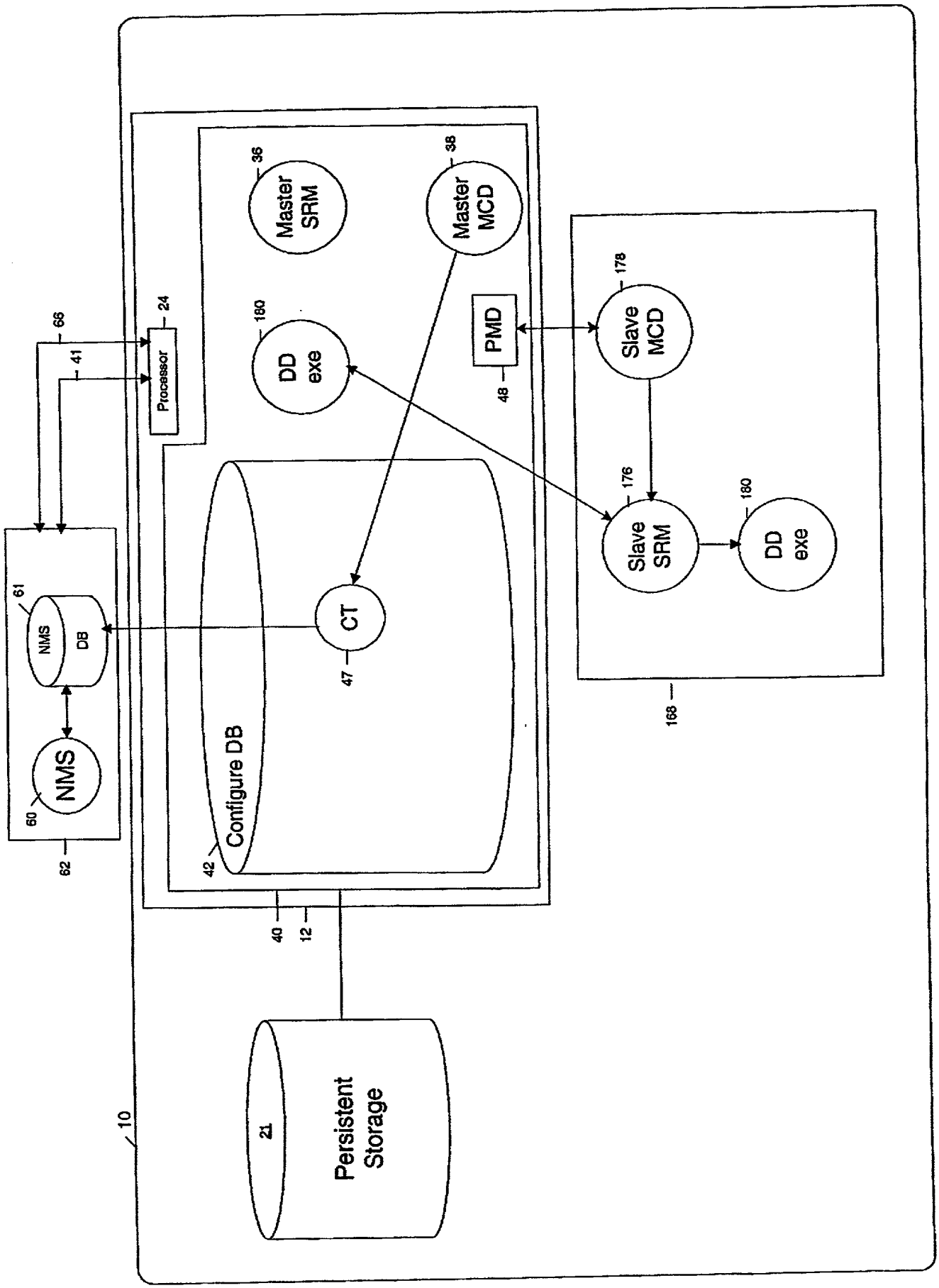

10
Persistent Storage
21
Master SRM
36
Master MCD
38
DD exe
180
PMD
48
Processor
24
66
41
Configure DB
42
CT
47
40
12
NMS
DB
61
NMS
60
62
Slave MCD
178
Slave SRM
176
DD exe
180
168

FIG. 19

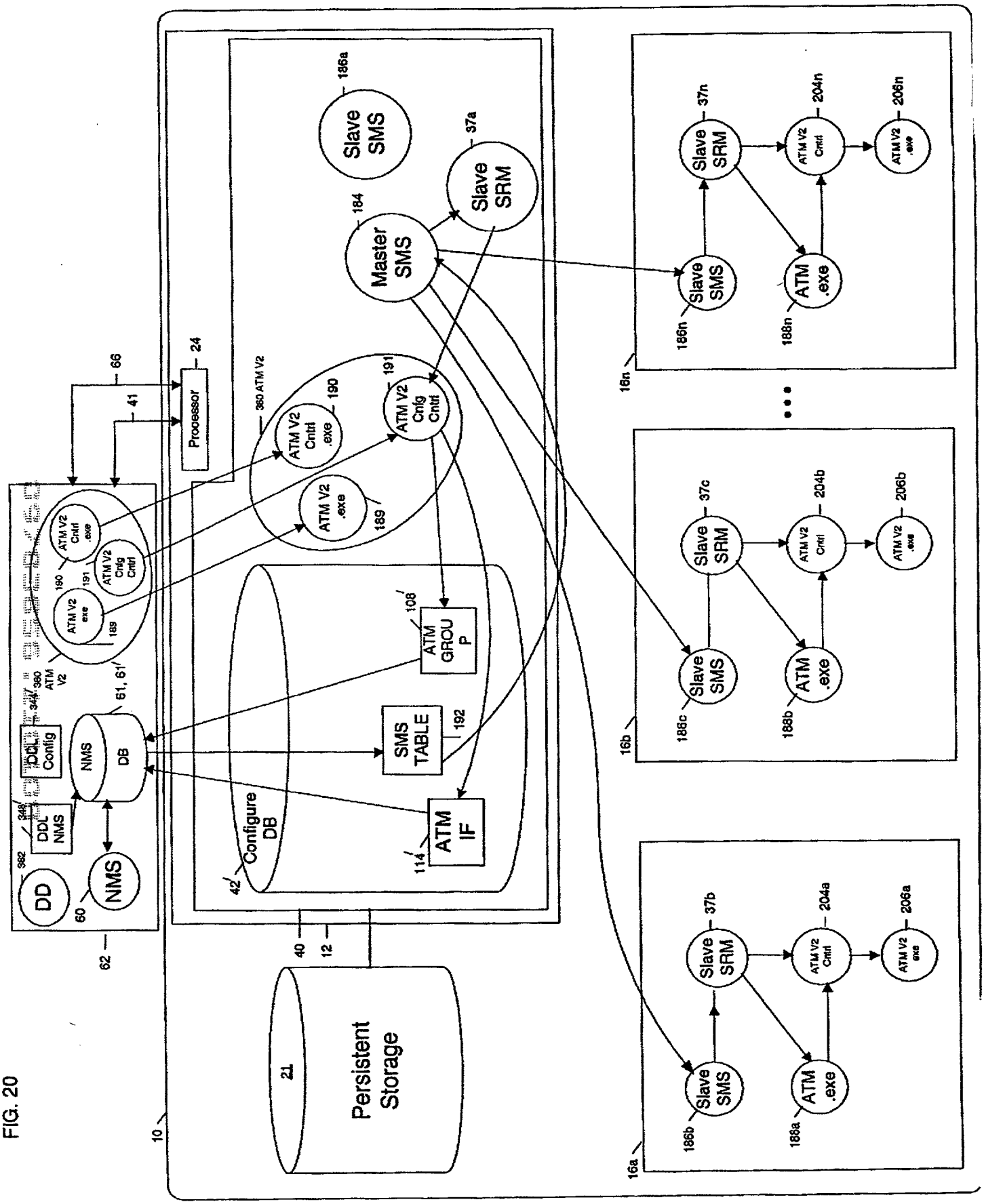

10
Persistent Storage
21
Configure DB
42
ATM IF
114
SMS TABLE
192
ATM GROUP
108
DB
NMS
61, 61'
DDL Config
DDL NMS
348
DD
362
NMS
60
62
Processor
24
66
41
40
12
ATM V2
360
ATM V2 Cntrl .exe
190
ATM V2 Config Cntrl
191
ATM V2 .exe
189
360 ATM V2
Master SMS
184
Slave SMS
186a
Slave SRM
37a
16a
Slave SMS
186b
ATM .exe
188a
Slave SRM
37b
ATM V2 Cntrl
204a
ATM V2 .exe
206a
16b
Slave SMS
186c
ATM .exe
188b
Slave SRM
37c
ATM V2 Cntrl
204b
ATM V2 .exe
206b
16n
Slave SMS
186n
ATM .exe
188n
Slave SRM
37n
ATM V2 Cntrl
204n
ATM V2 .exe
206n

FIG. 20

REPLACEMENT SHEET

FIG. 21

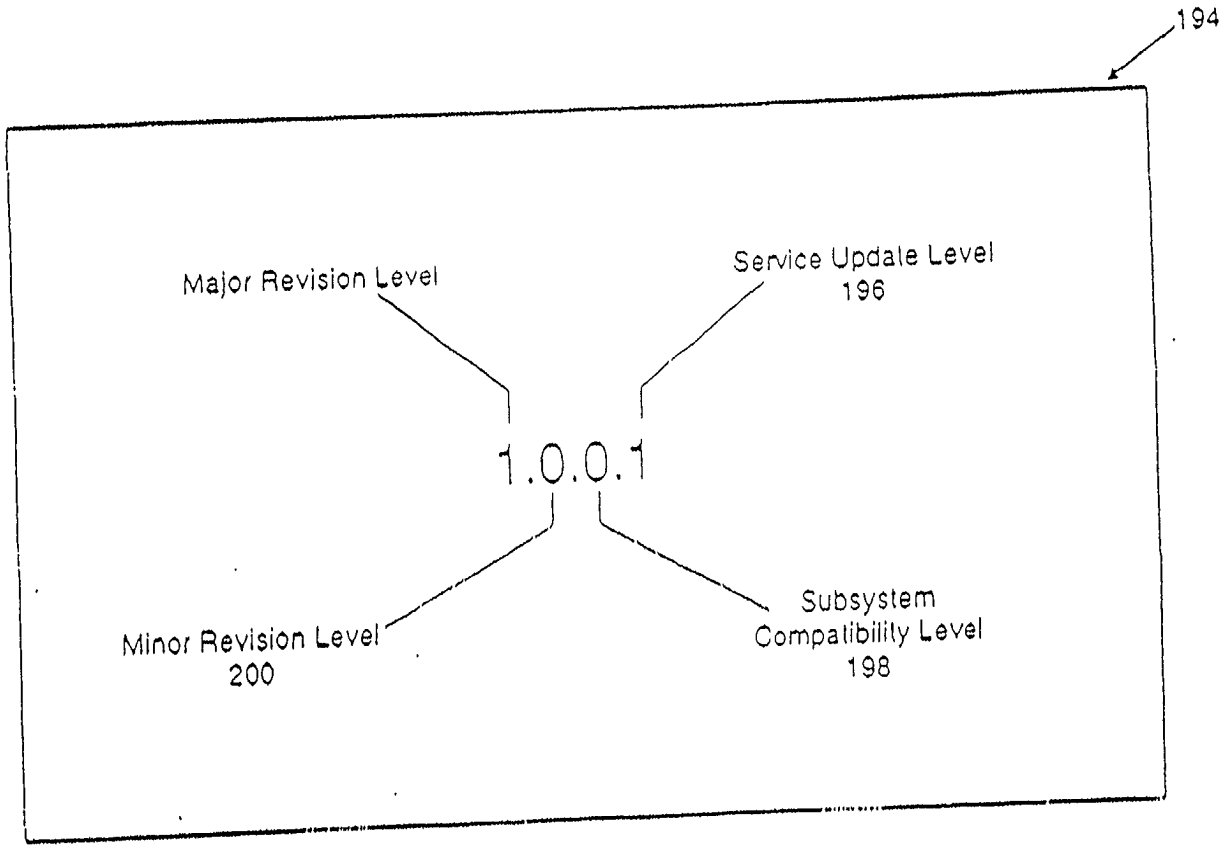
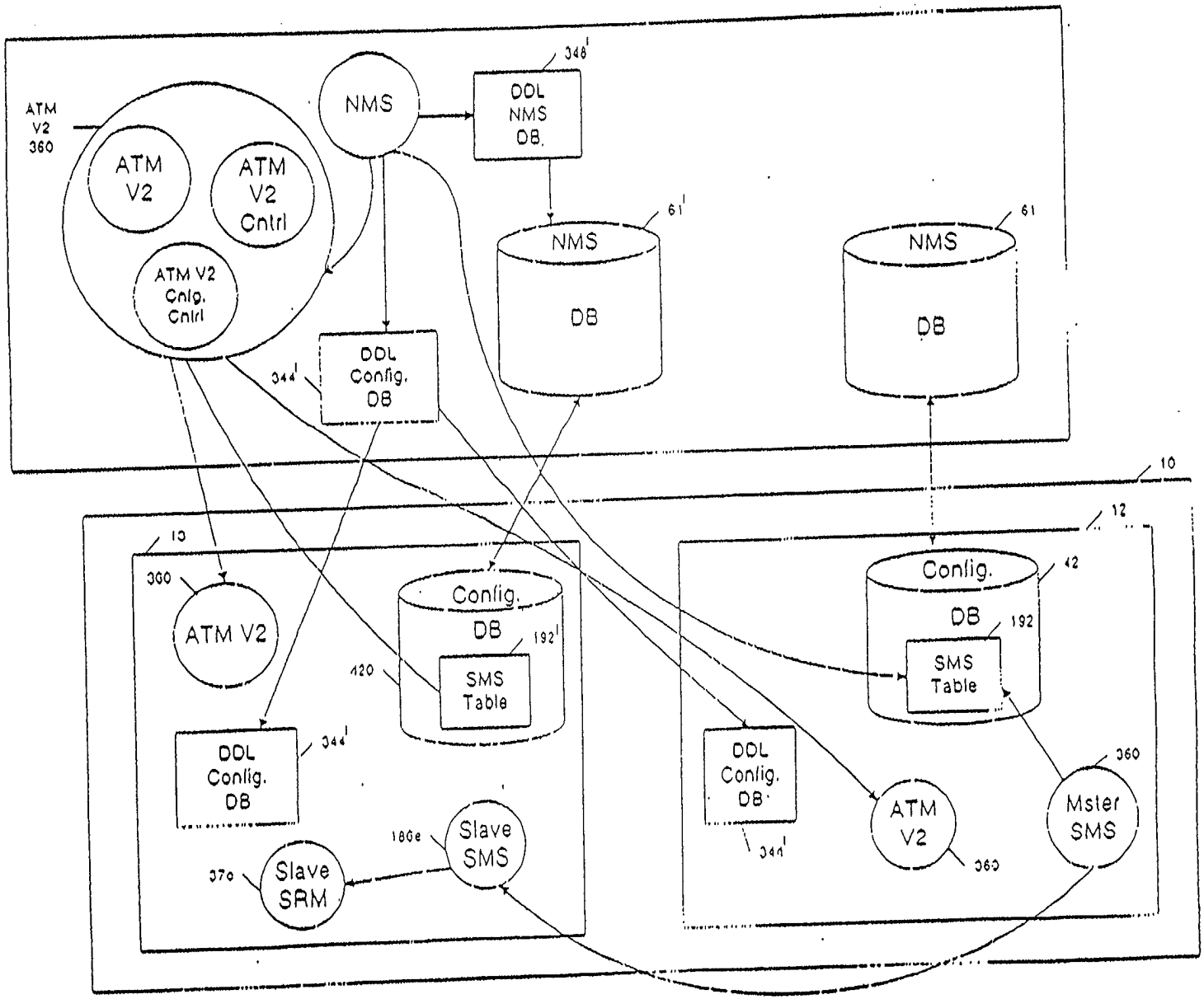


FIG. 22


ATM V2 360
ATM V2
ATM V2 Cntrl
ATM V2 Config. Cntrl
NMS
348'
DDL NMS DB.
61'
NMS
DB
61
NMS
DB
344'
DDL Config. DB
10
10
360
ATM V2
Config.
DB
192'
SMS Table
420
DDL Config. DB
344'
Slave SMS
180e
Slave SRM
370
12
Config.
42
DB
192
SMS Table
DDL Config. DB
344'
ATM V2
365
360
Mster SMS

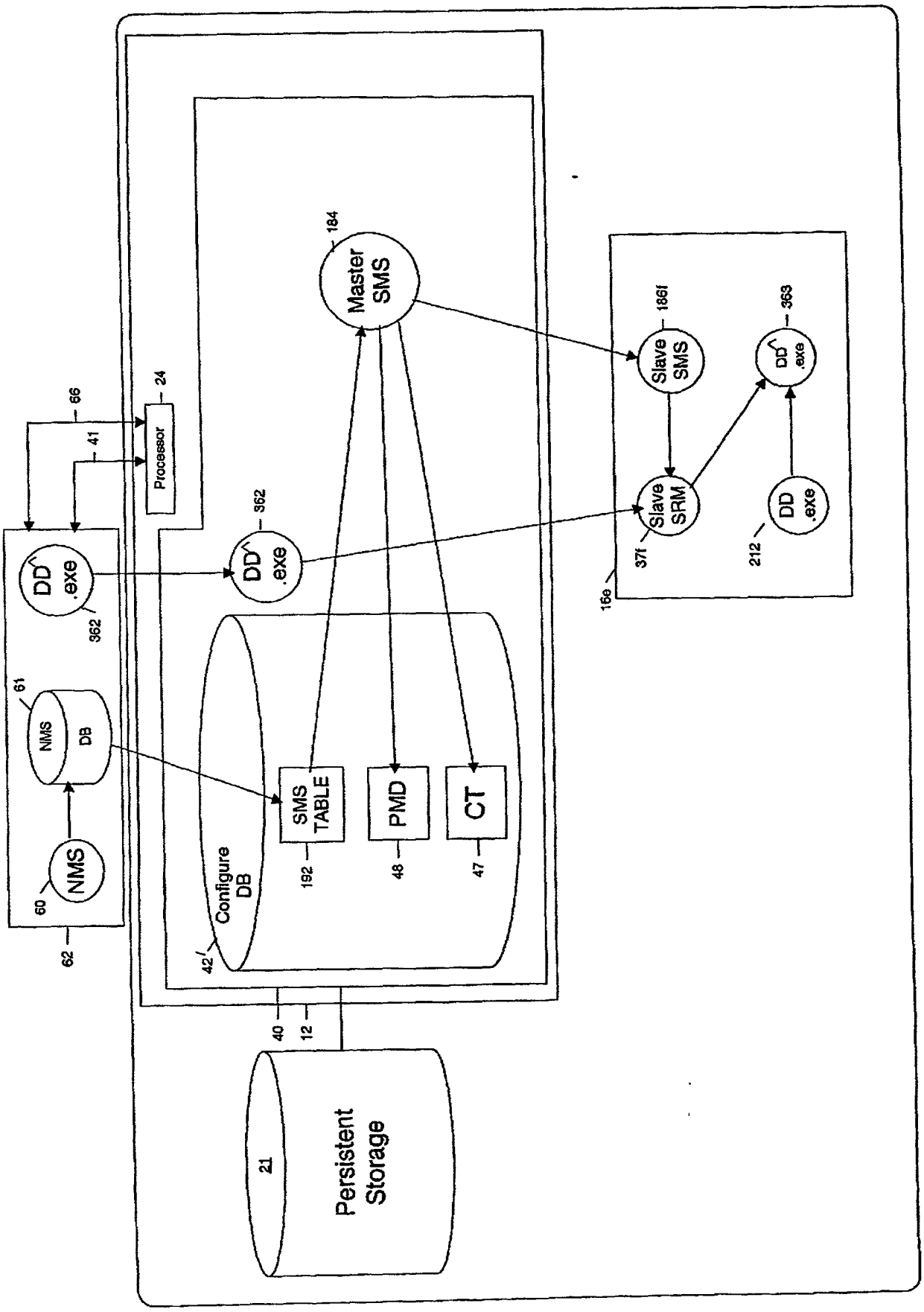

Master SMS
184
Slave SMS
186f
DD .exe
363
Slave SRM
37f
DD .exe
212
16e
Processor
24
66
41
DD .exe
362
362
NMS
DB
61
NMS
60
62
Configure DB
42
SMS TABLE
192
PMD
48
CT
47
40
12
Persistent Storage
21

FIG. 23

FIG. 24

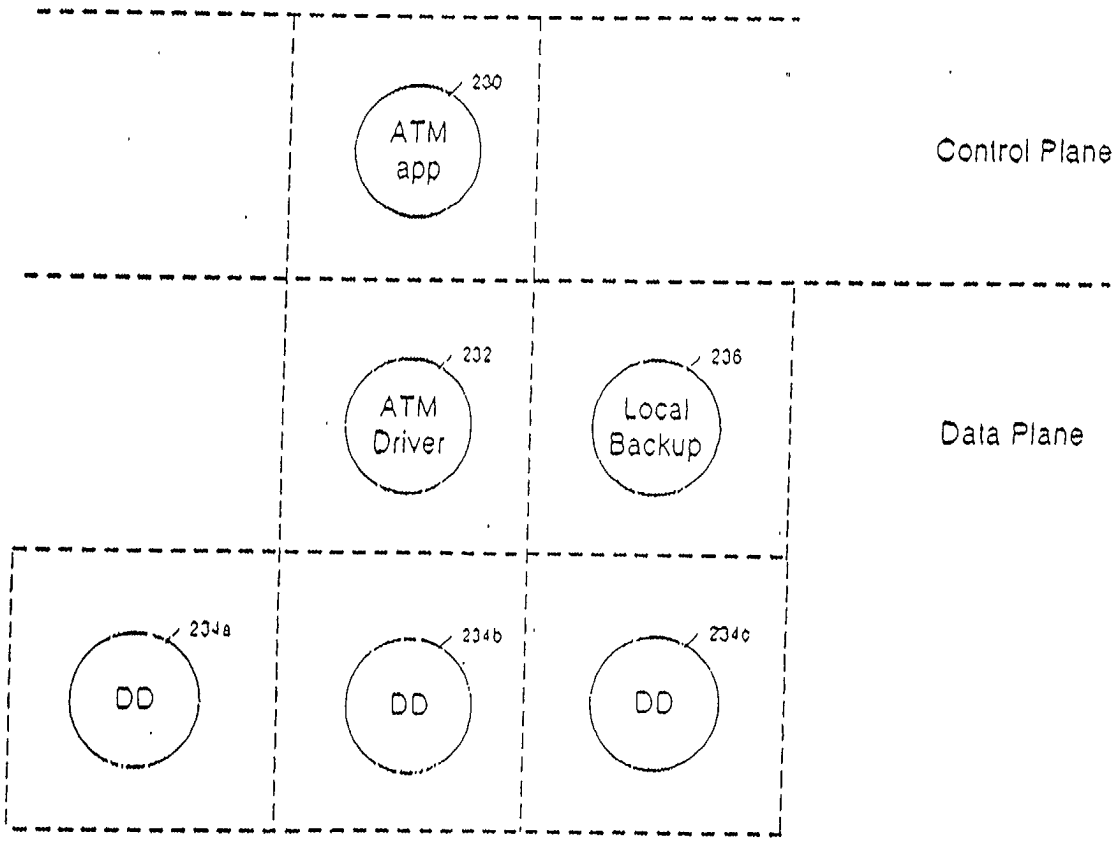
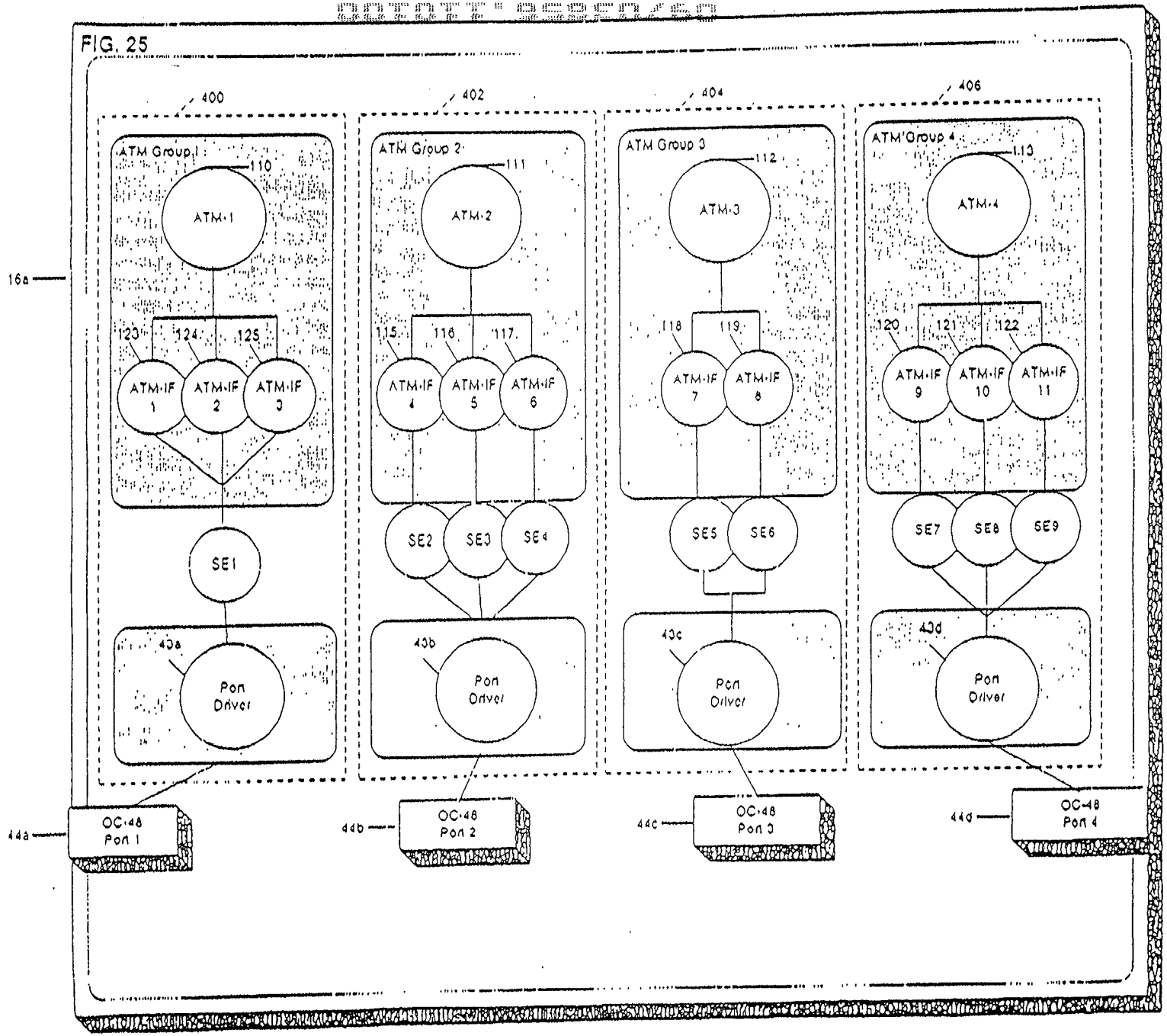

230
ATM app
Control Plane
232
ATM Driver
236
Local Backup
Data Plane
234a
DD
234b
DD
234c
DD

FIG. 25


400
402
404
406
ATM Group 1
ATM Group 2
ATM Group 3
ATM Group 4
110
111
112
113
ATM-1
ATM-2
ATM-3
ATM-4
120
121
122
123
124
125
115
116
117
118
119
ATM-IF 1
ATM-IF 2
ATM-IF 3
ATM-IF 4
ATM-IF 5
ATM-IF 6
ATM-IF 7
ATM-IF 8
ATM-IF 9
ATM-IF 10
ATM-IF 11
SE1
SE2
SE3
SE4
SE5
SE6
SE7
SE8
SE9
43a
43b
43c
43d
Pon Driver
16a
44a
44b
44c
44d
OC-48 Pon 1
OC-48 Pon 2
OC-48 Pon 3
OC-48 Pon 4

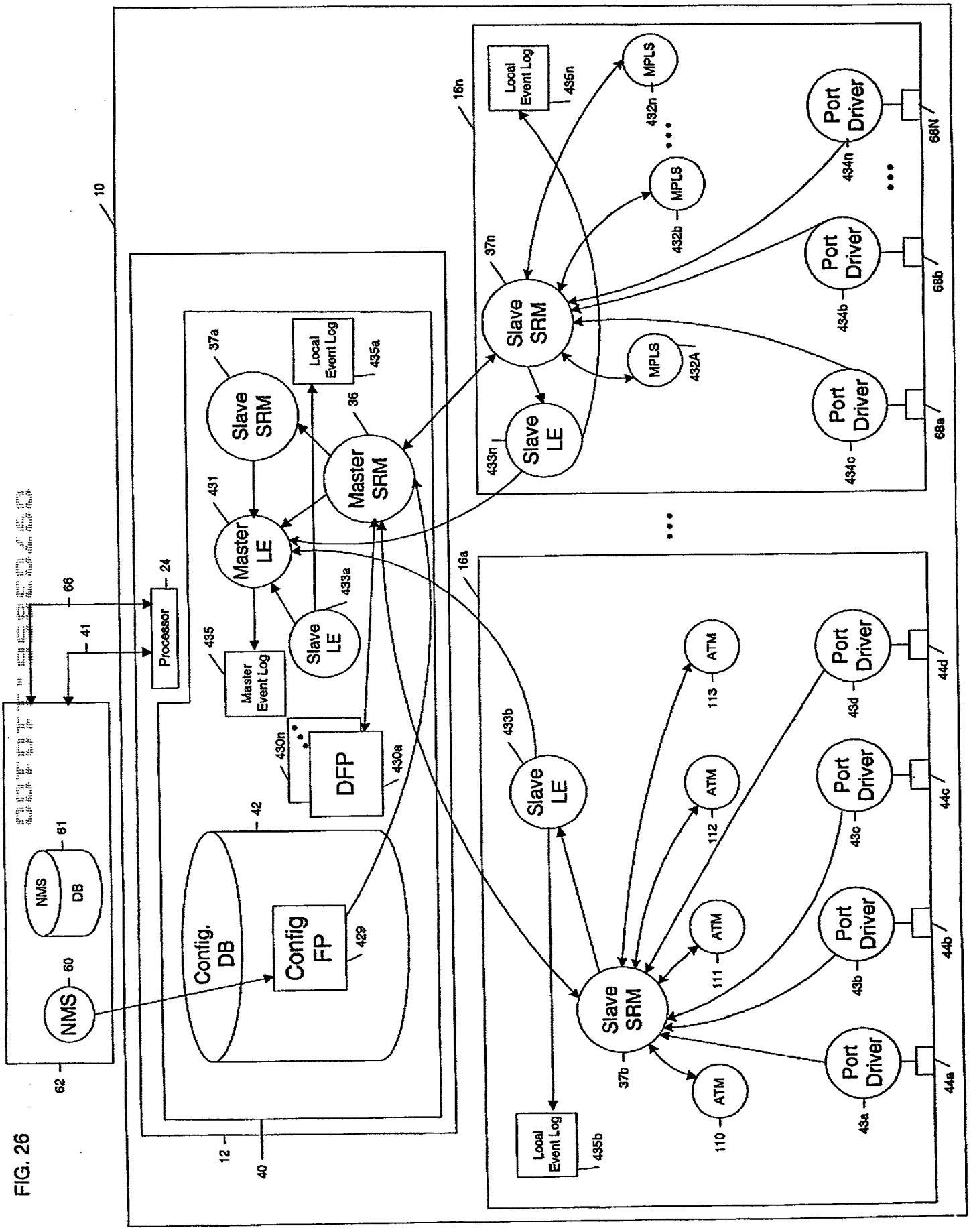

10
12
40
62
NMS
60
NMS DB
61
66
41
24
Processor
42
Config. DB
Config FP
429
DFP
430a
430n
Master Event Log
435
Master LE
431
Master SRM
36
Slave SRM
37a
Slave LE
433a
Local Event Log
435a
16a
Slave SRM
37b
Slave LE
433b
Local Event Log
435b
ATM
110
ATM
111
ATM
112
ATM
113
Port Driver
43a
Port Driver
43b
Port Driver
43c
Port Driver
43d
44a
44b
44c
44d
16n
Slave SRM
37n
Slave LE
433n
Local Event Log
435n
MPLS
432A
MPLS
432b
MPLS
432n
Port Driver
434o
Port Driver
434b
Port Driver
434n
68a
68b
68N

FIG. 26

FIG. 27

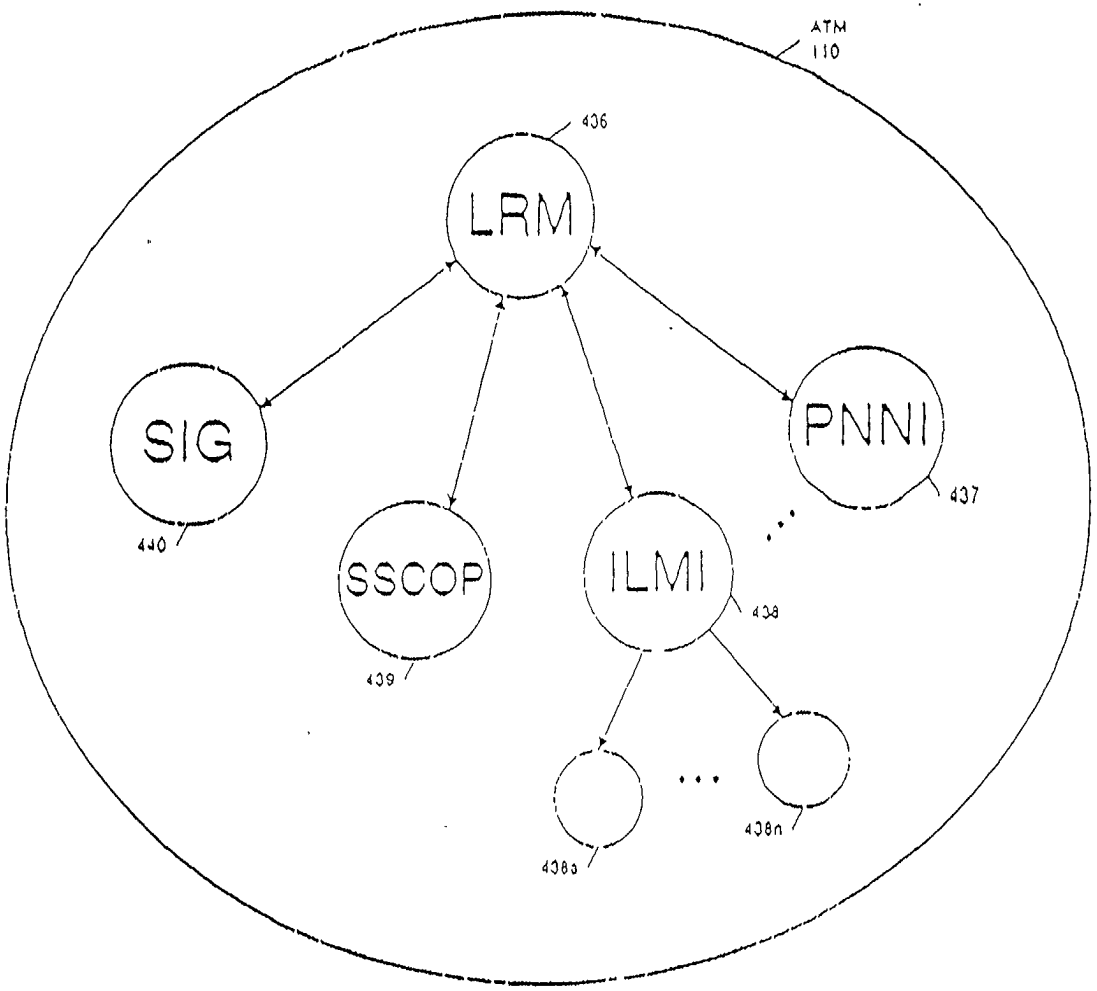
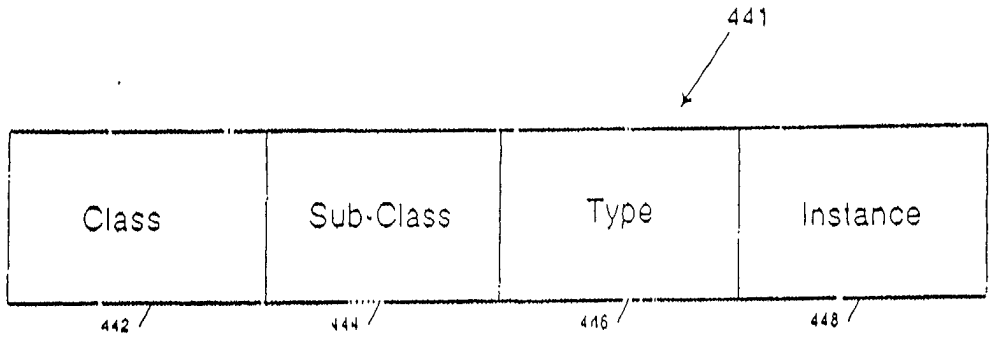


FIG. 28



PATENT PENDING

FIG. 29

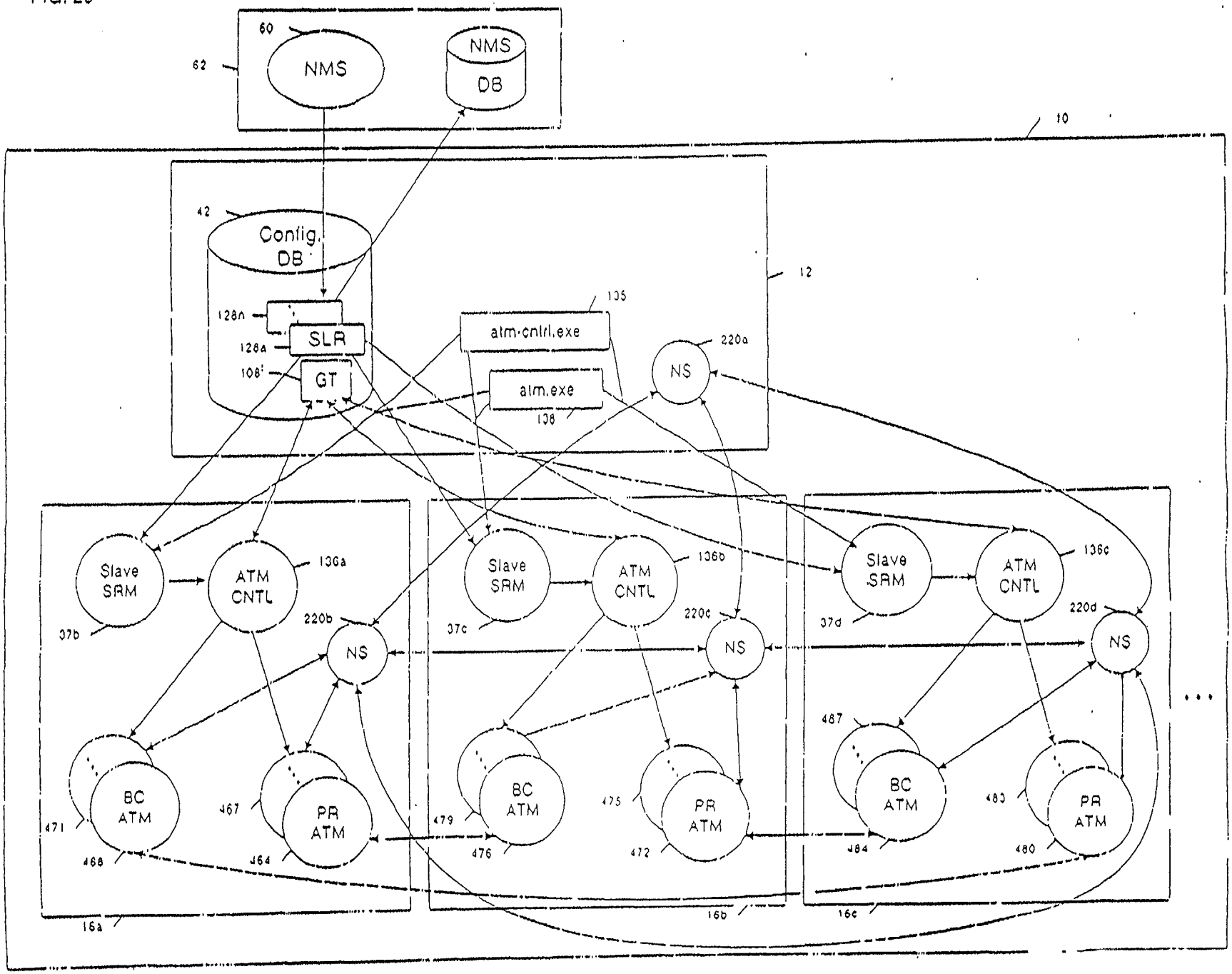


FIG.30

Group Table 108

	Group #	Primary Card LID (447)	Backup Card LID (449)	...
450	1	30	31	
451	2	30	31	
452	3	30	31	
453	4	30	31	
454	5	31	32	
455	6	31	32	
456	7	31	32	
457	8	31	32	
458	9	32	30	
459	10	32	30	
460	11	32	30	
461	12	32	30	
	⋮	⋮	⋮	⋮

Fig. 31a

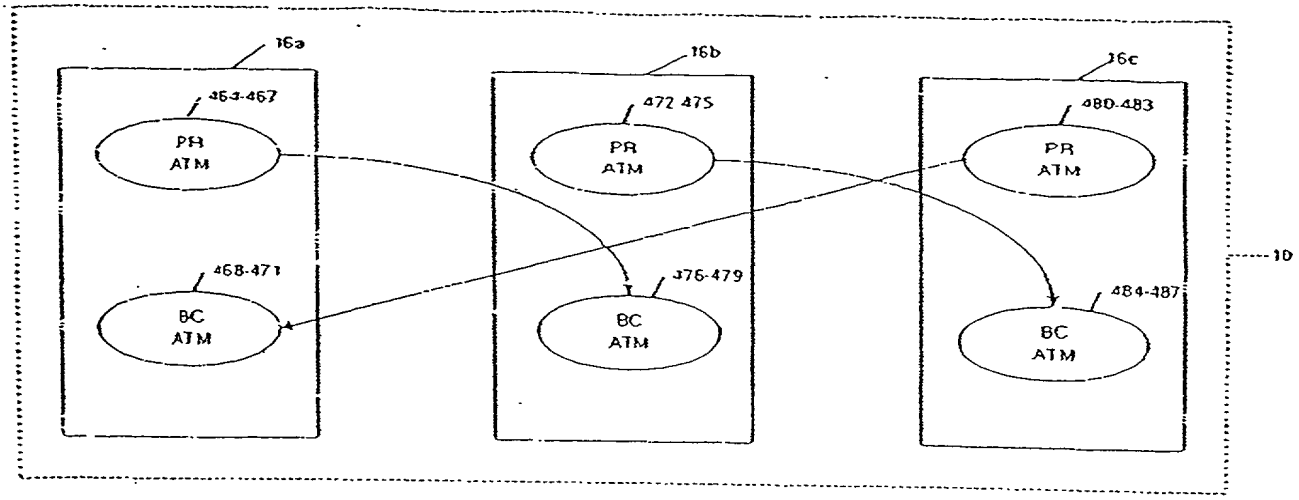


Fig. 31b

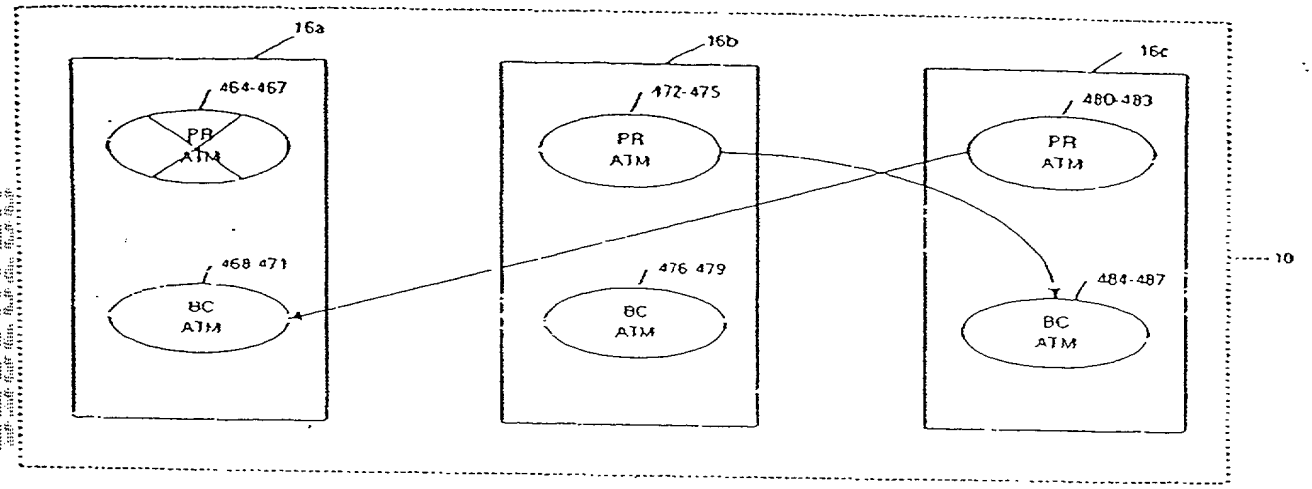


Fig. 31c

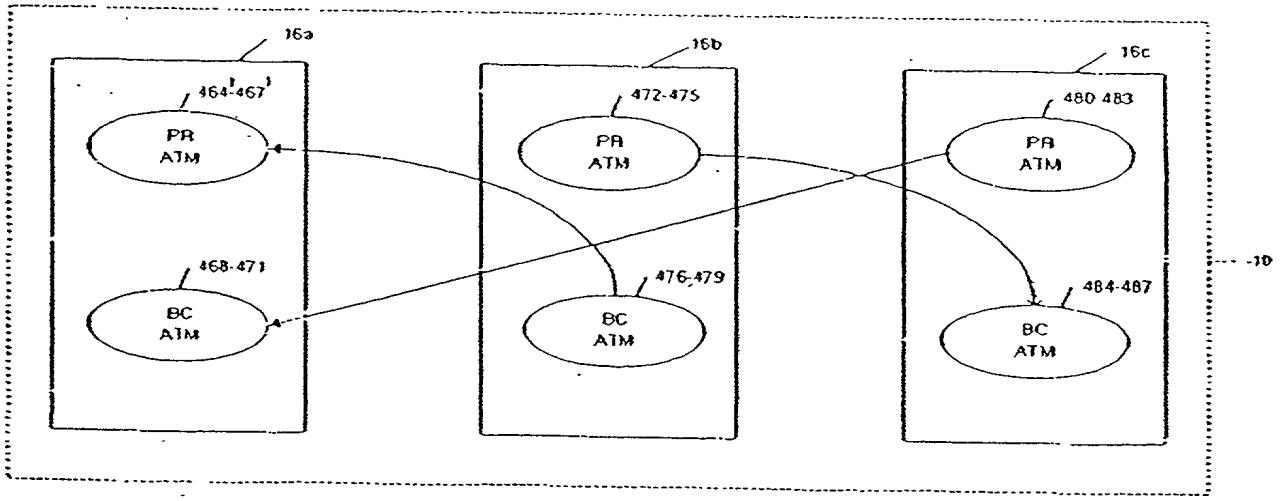


Fig. 32d

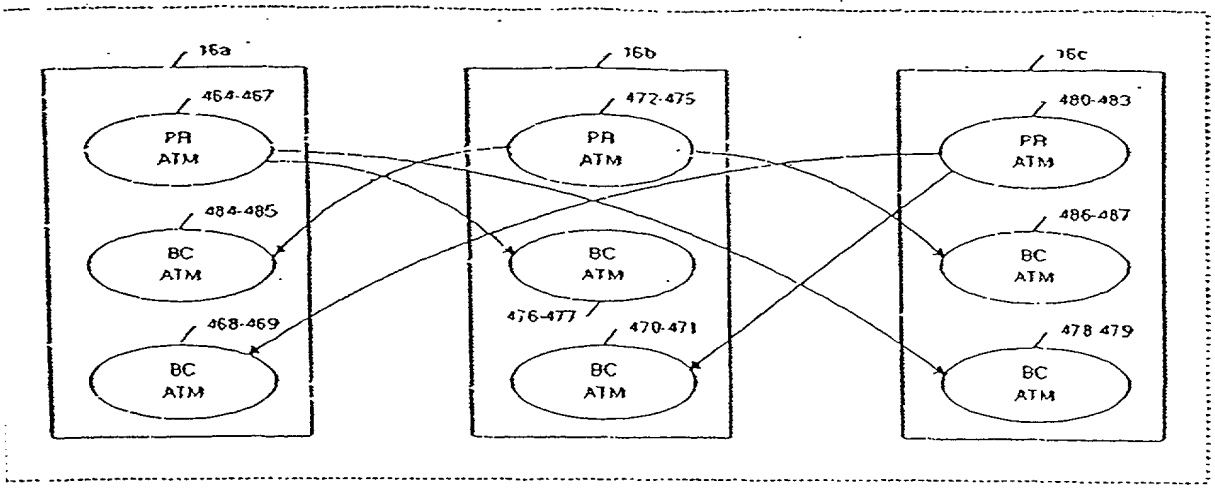


Fig. 32A

Fig. 32b

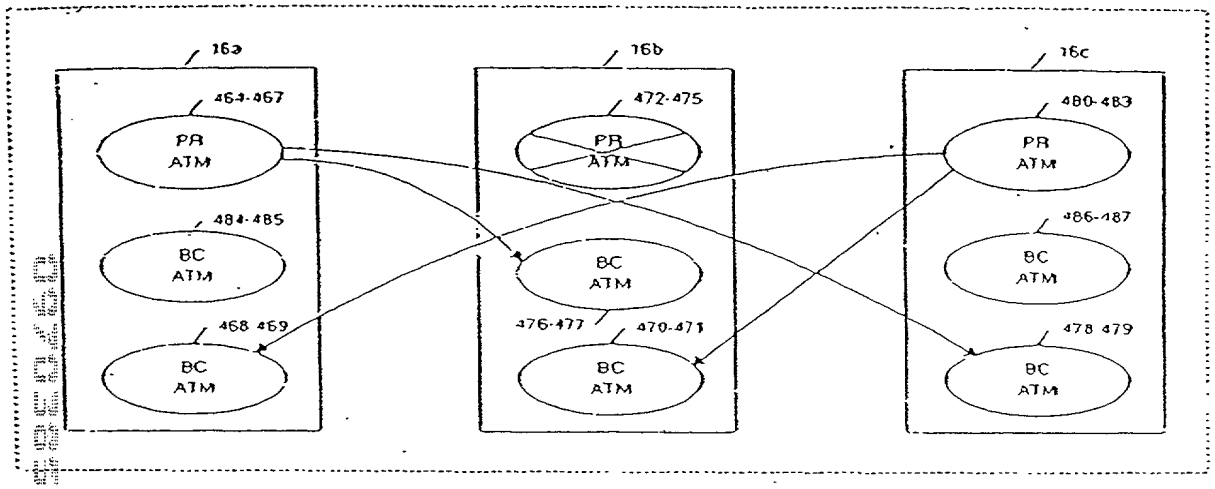


Fig. 32c

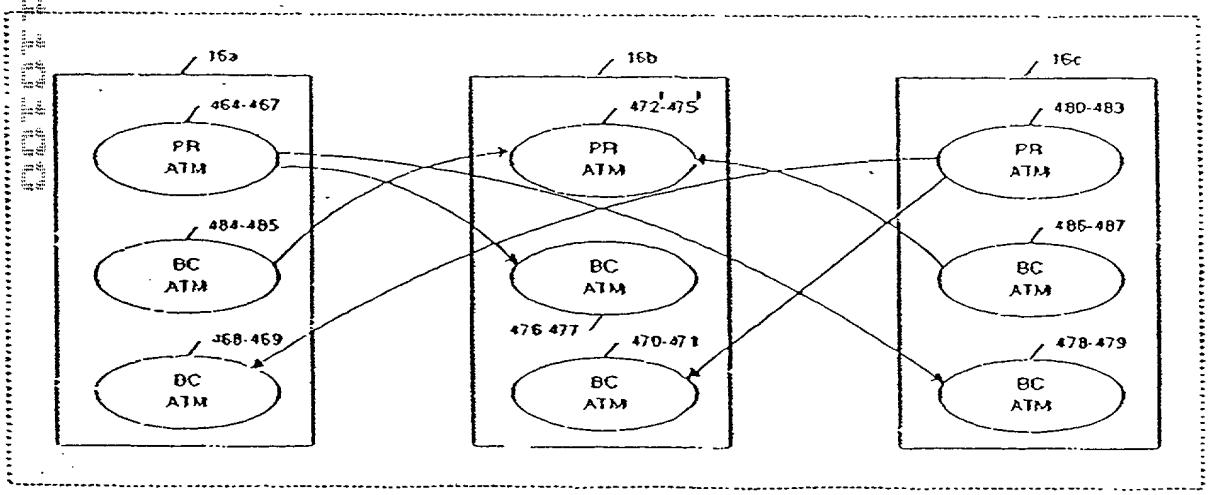


FIG. 33a

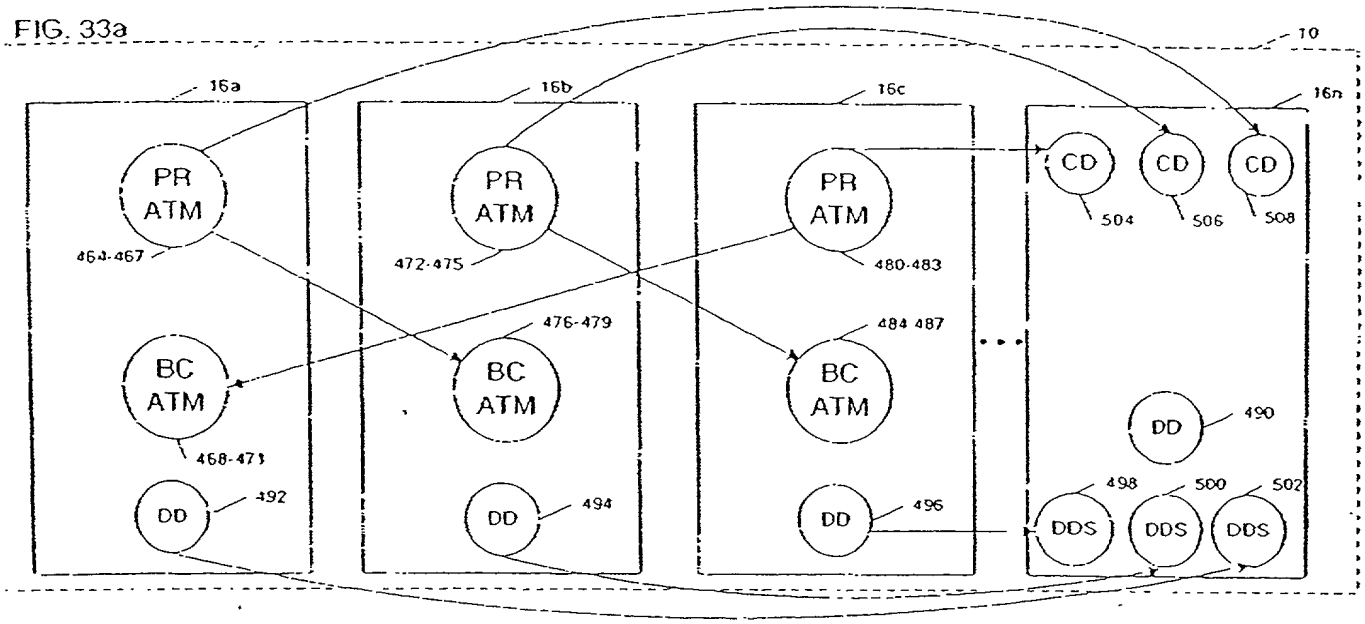


FIG. 33b

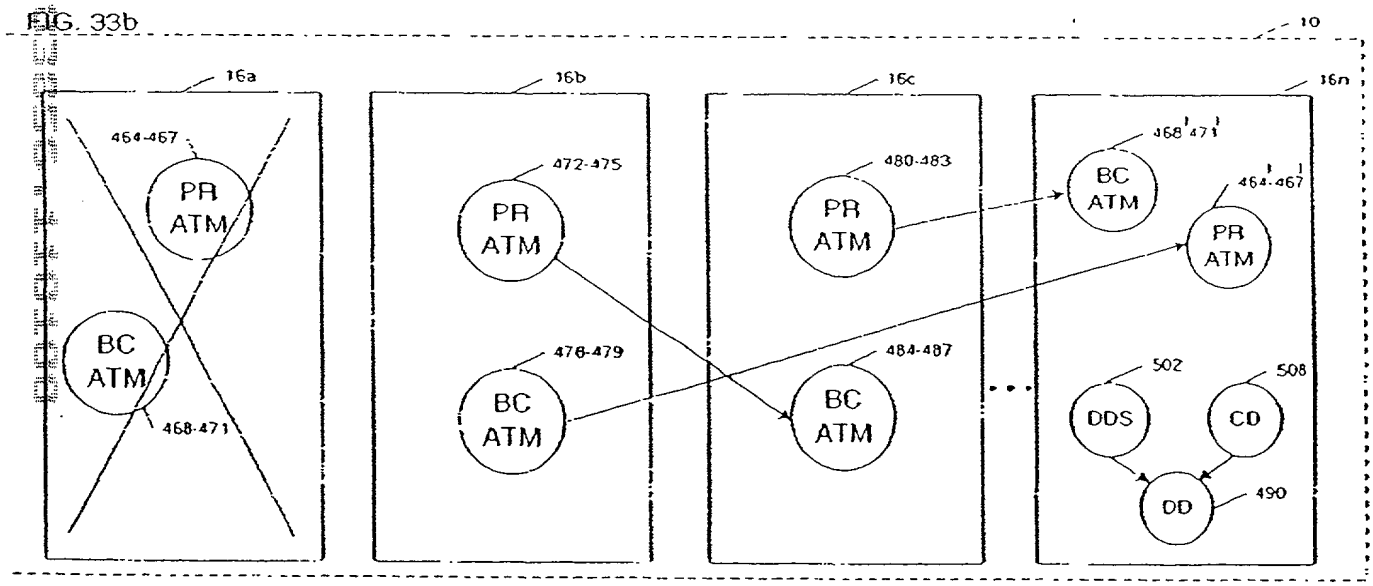


FIG. 33c

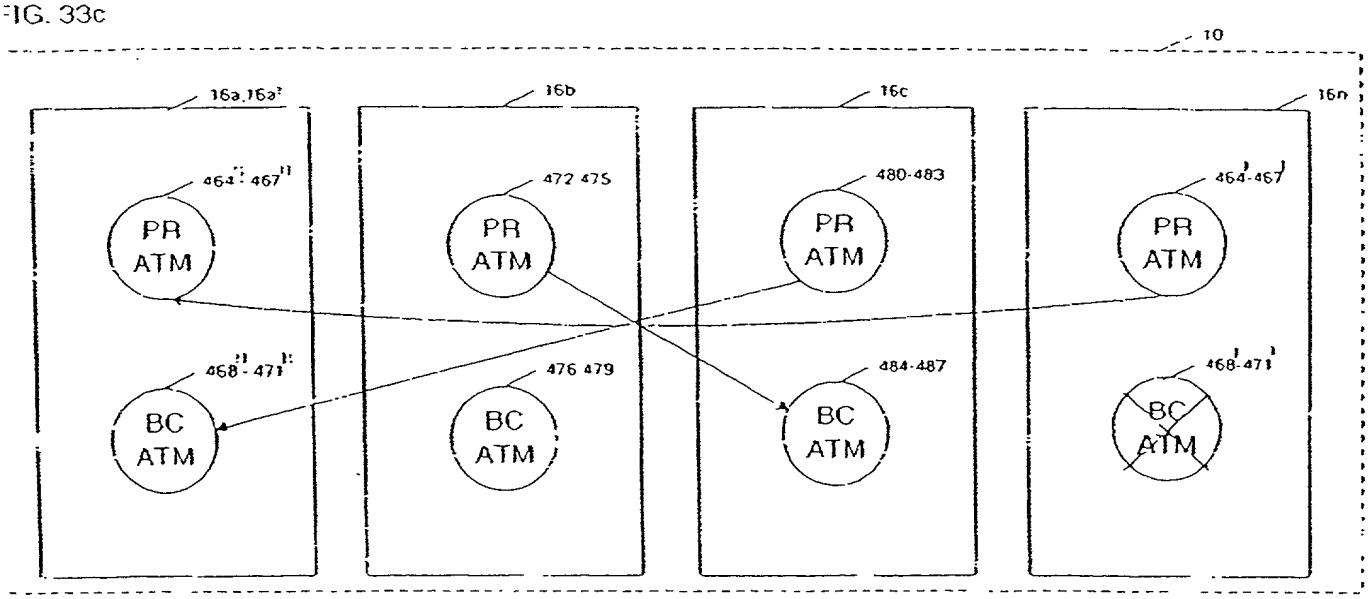


FIG. 33d

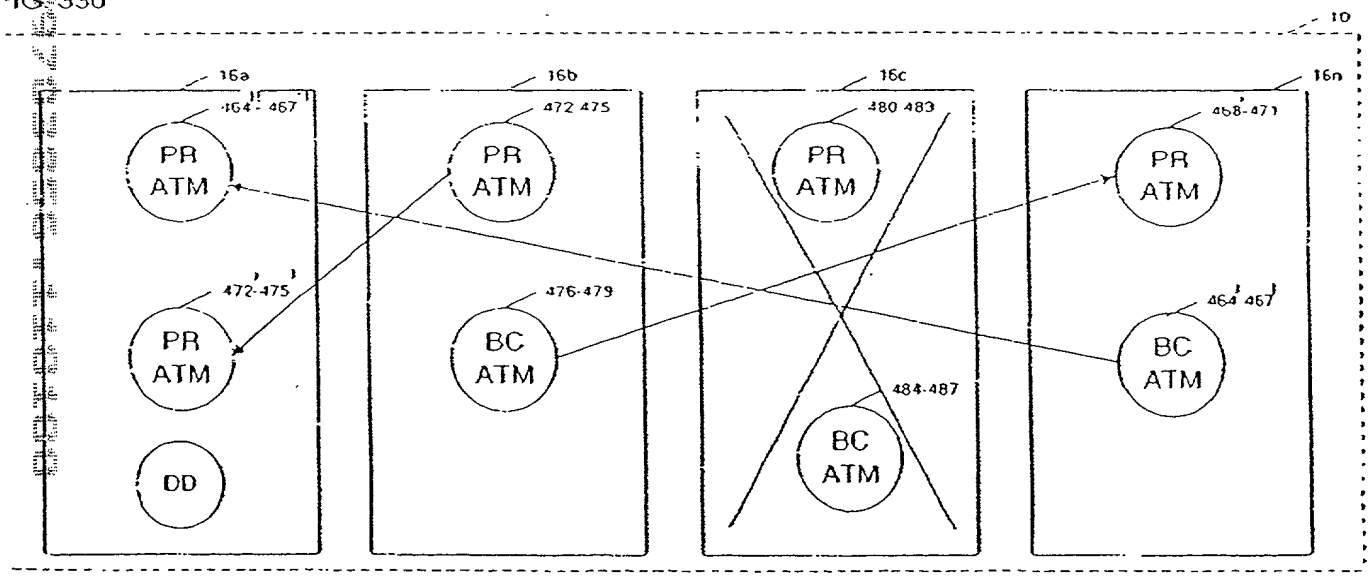


Fig. 34a

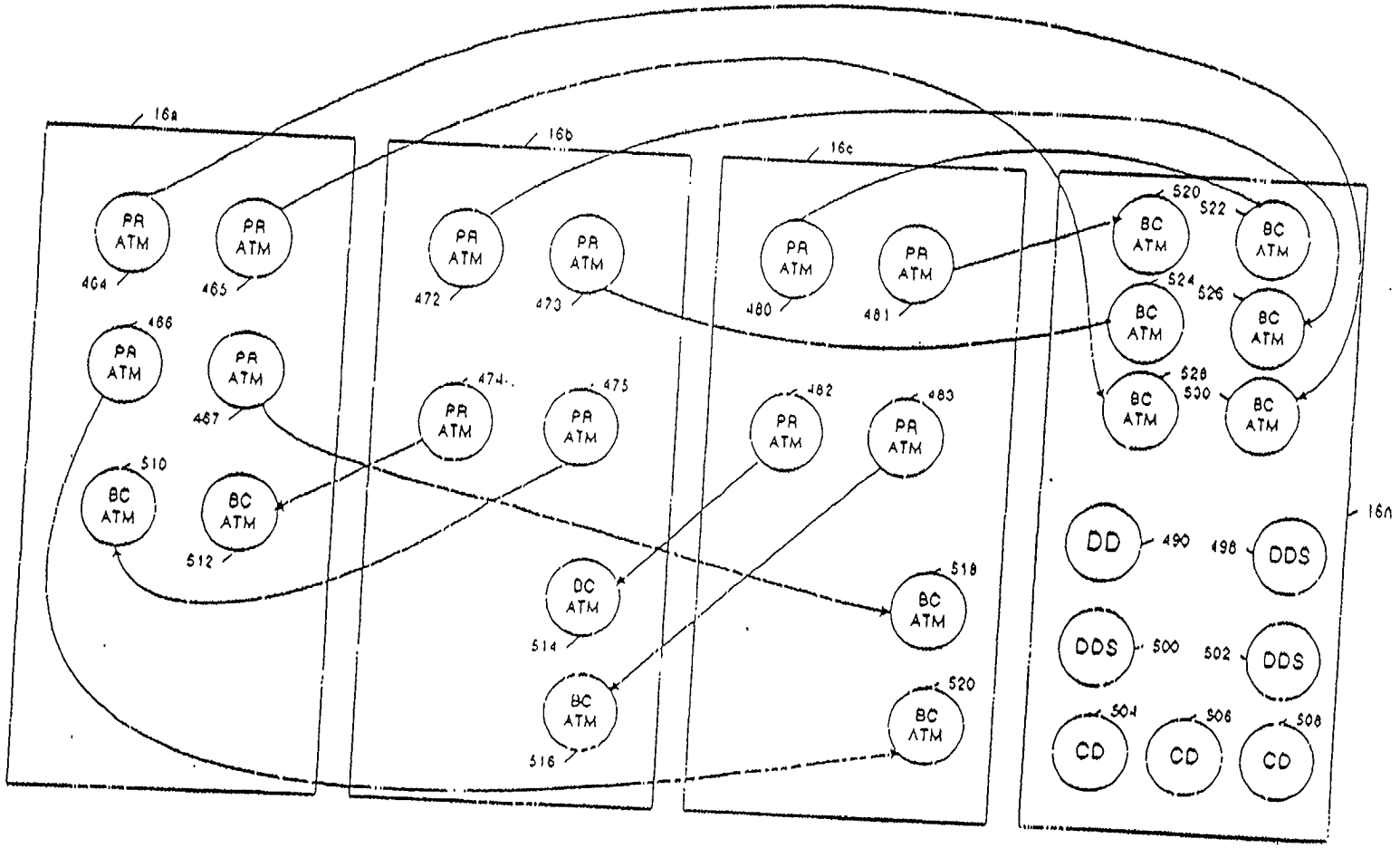
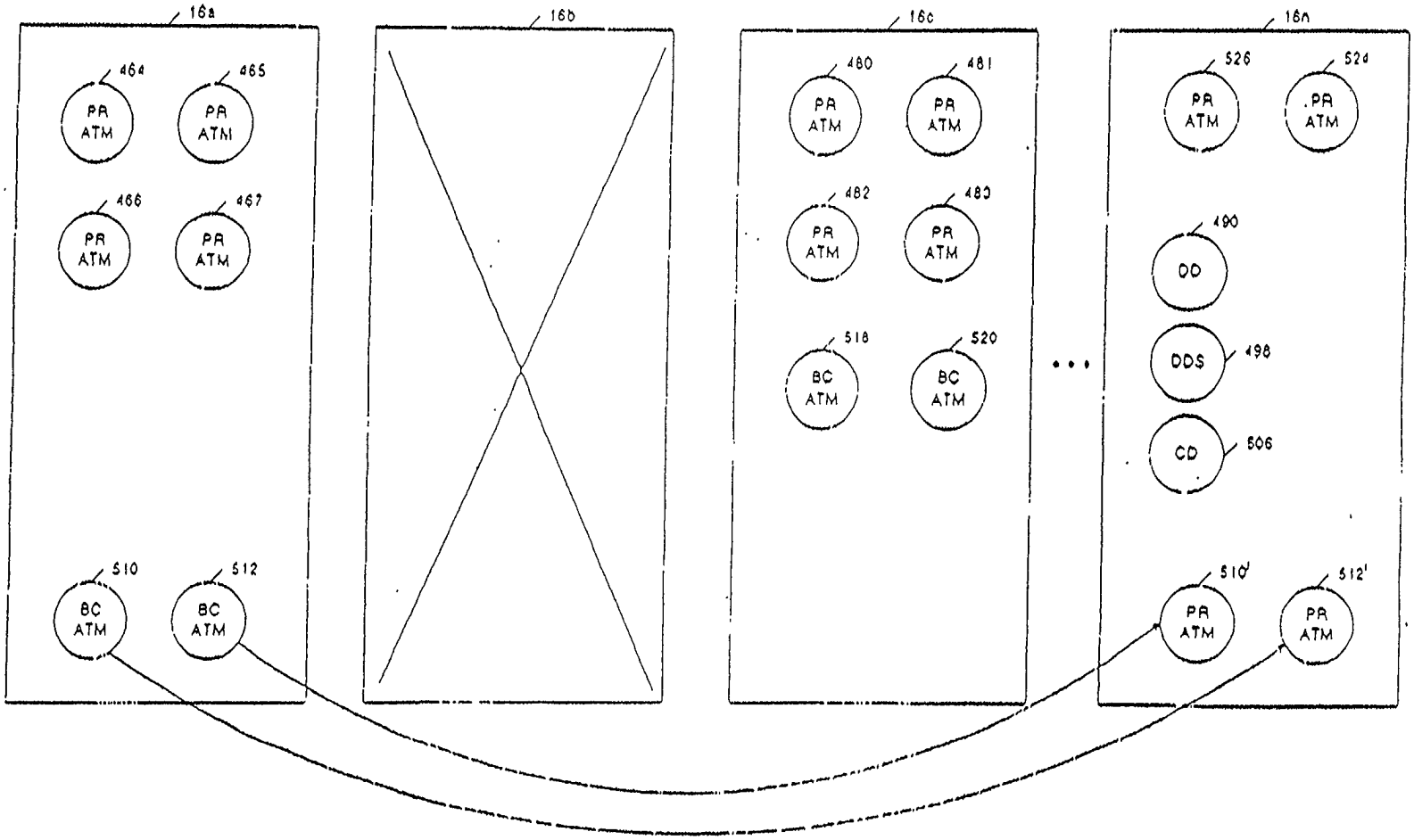

16a
16b
16c
16n
PR ATM
464
465
472
473
480
481
466
467
474
475
482
483
BC ATM
510
512
514
516
518
520
522
524
526
528
530
DD
490
498
DDS
500
502
CD
504
506
508

FIG. 34b


16a
16b
16c
16n
464
PR ATM
465
PR ATM
466
PR ATM
467
PR ATM
510
BC ATM
512
BC ATM
480
PR ATM
481
PR ATM
482
PR ATM
483
PR ATM
518
BC ATM
520
BC ATM
526
PR ATM
52d
PR ATM
490
DD
DDS
498
CD
506
510'
PR ATM
512'
PR ATM

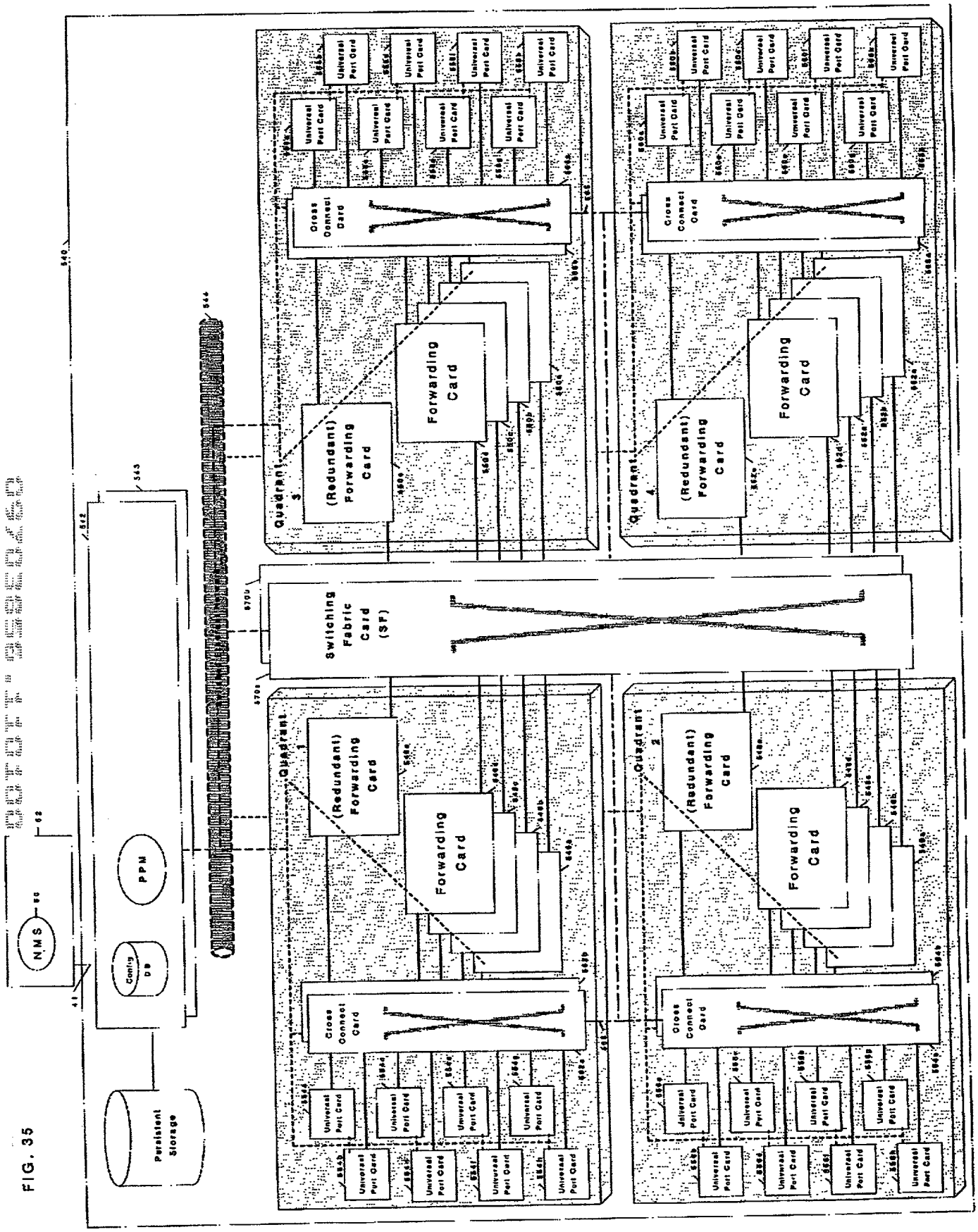
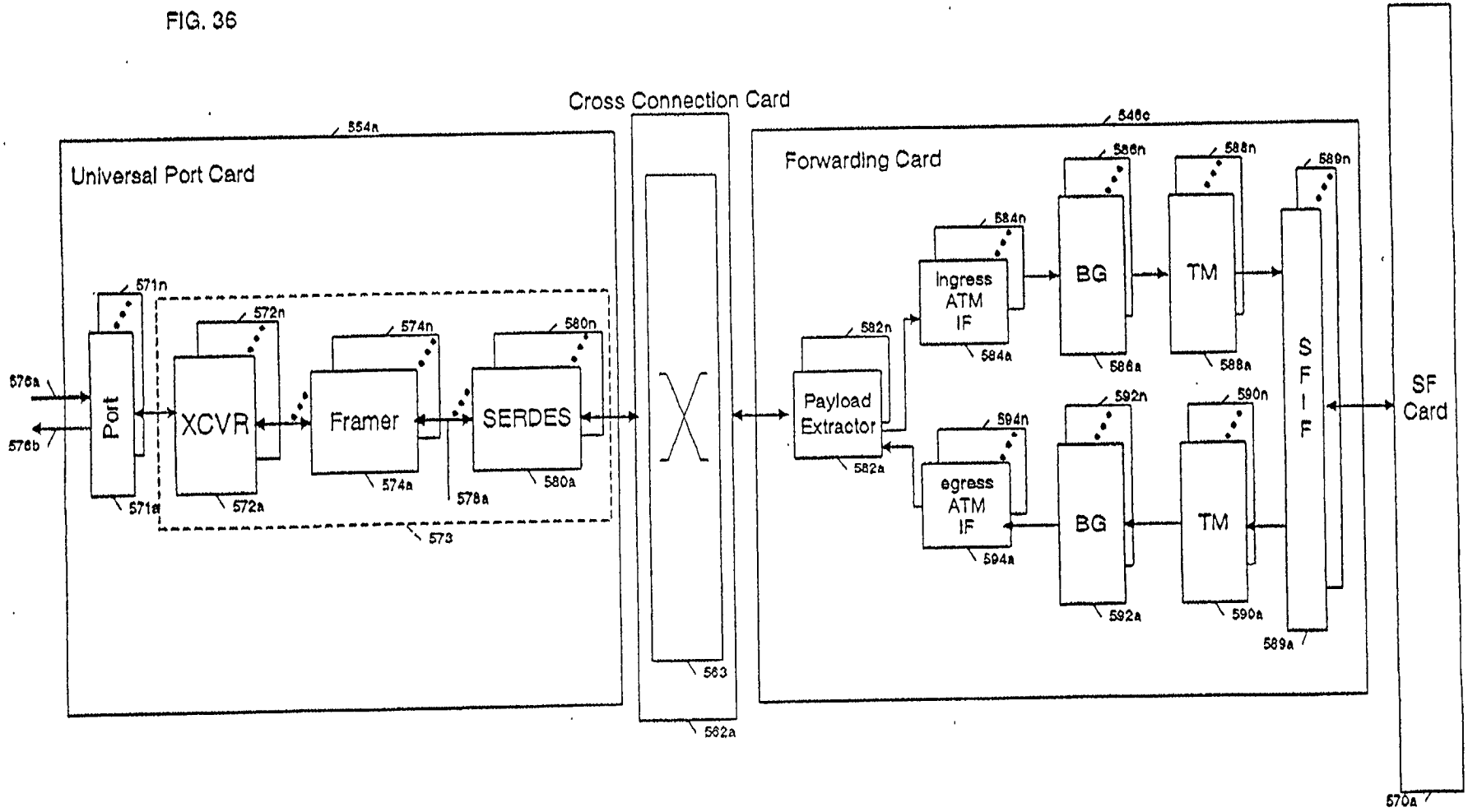

NMS
PPM
Config DB
Persistent Storage
Switching Fabric Card (SF)
Quadrant 1
Quadrant 2
Quadrant 3
Quadrant 4
(Redundant) Forwarding Card
Forwarding Card
Cross Connect Card
Universal Port Card

FIG. 35

PATENT PENDING

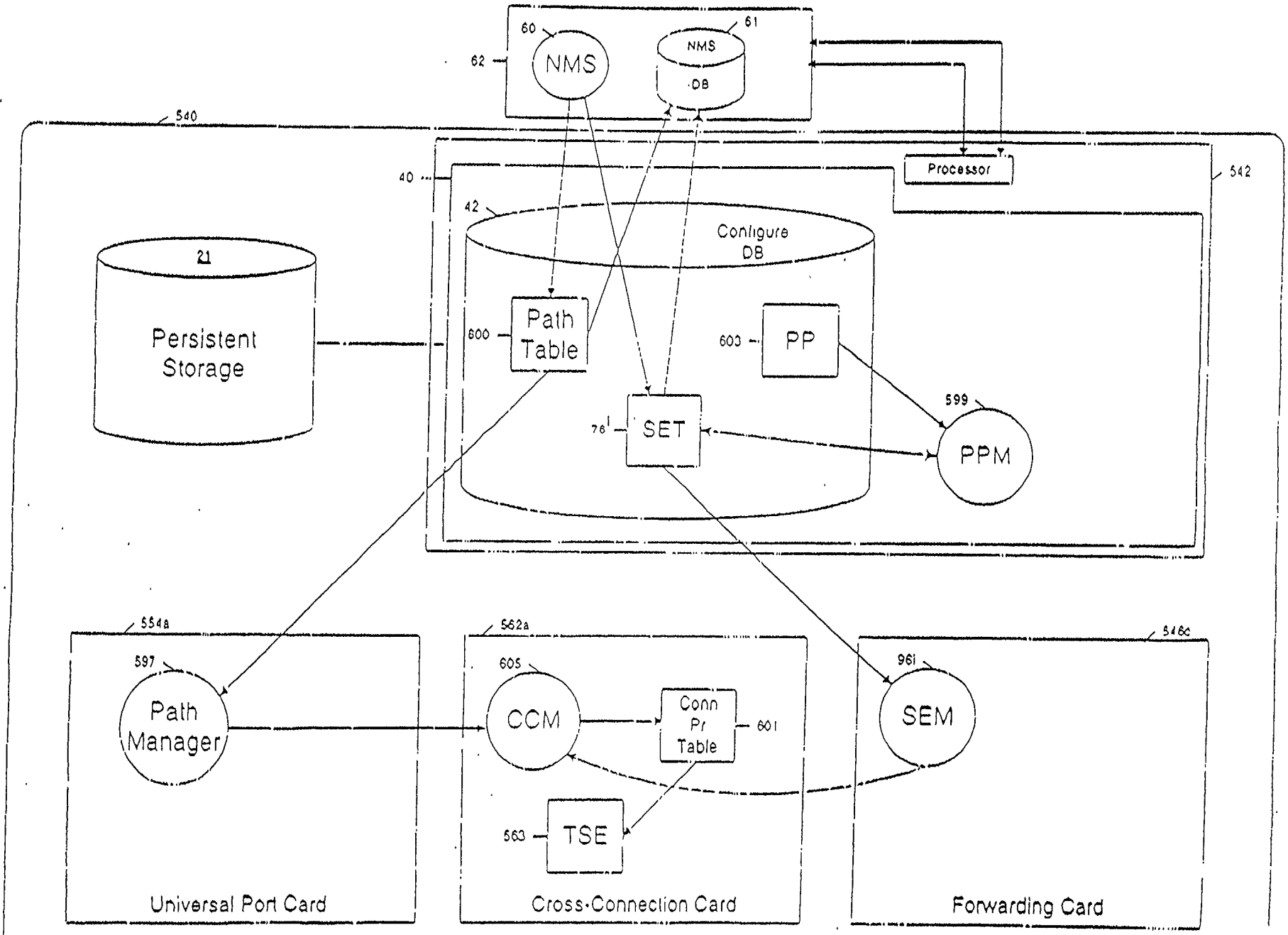
By: Equipe Communications; 781 795 2031; Jul-12-00 3:56PM; Page 2

FIG. 36



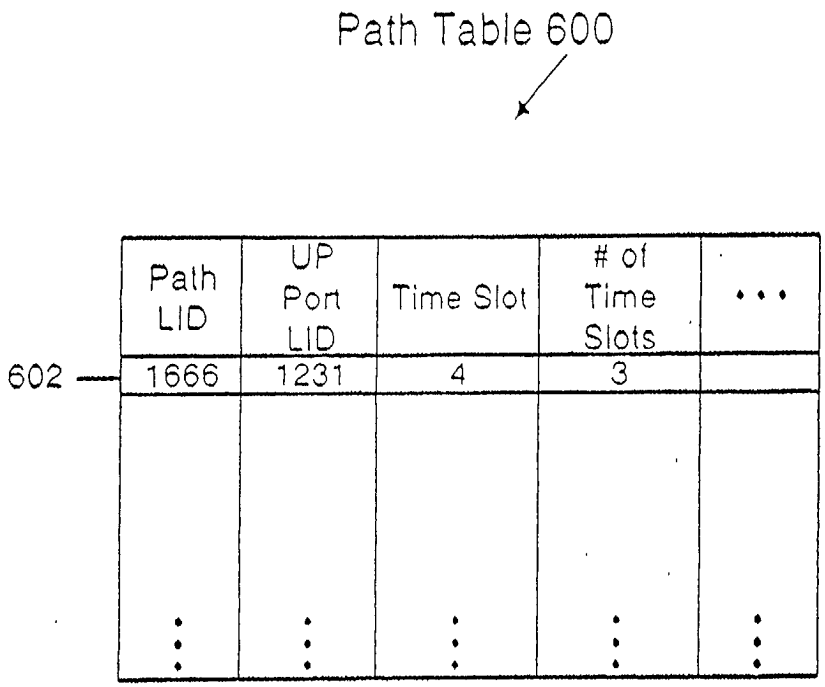
PATENT PENDING

FIG. 37



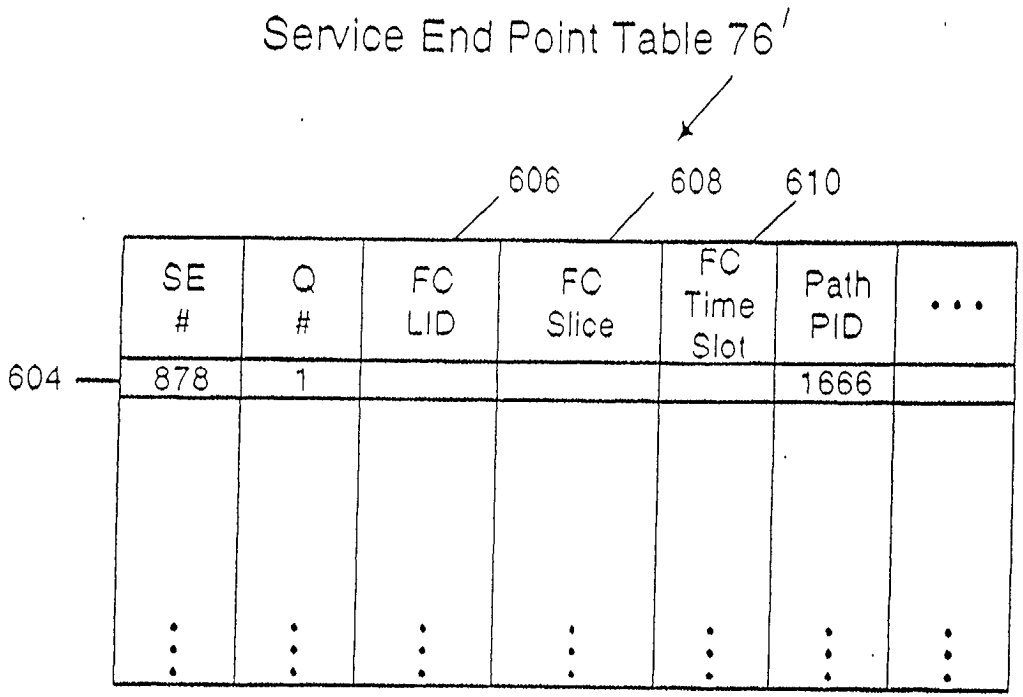
PATENT PENDING

FIG. 38



PATENT PENDING

FIG. 39



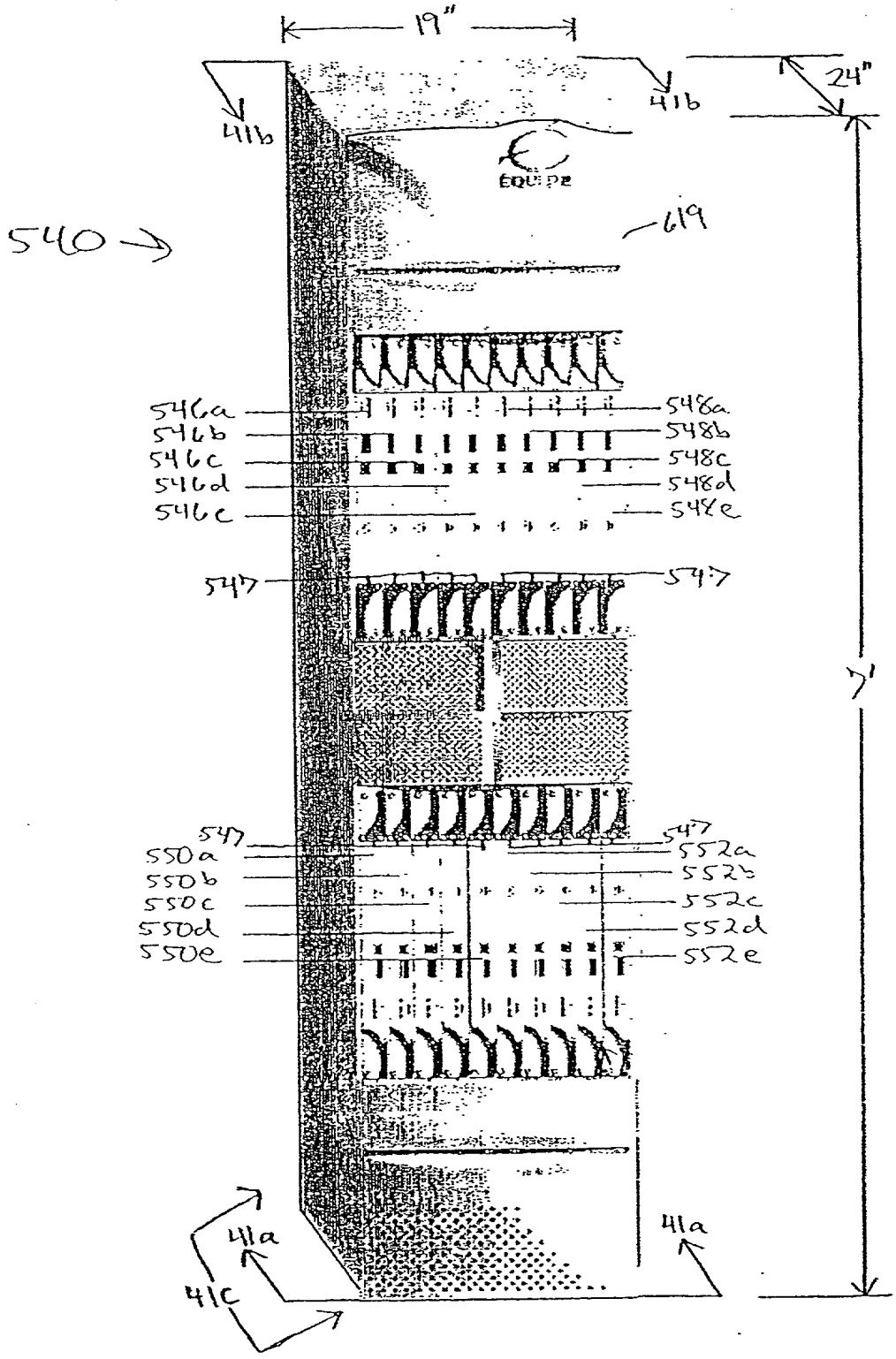


Fig. 40

Sent By: Equipe Communications; 781 795 2031; Jul-6-00 4:25PM; Page 2

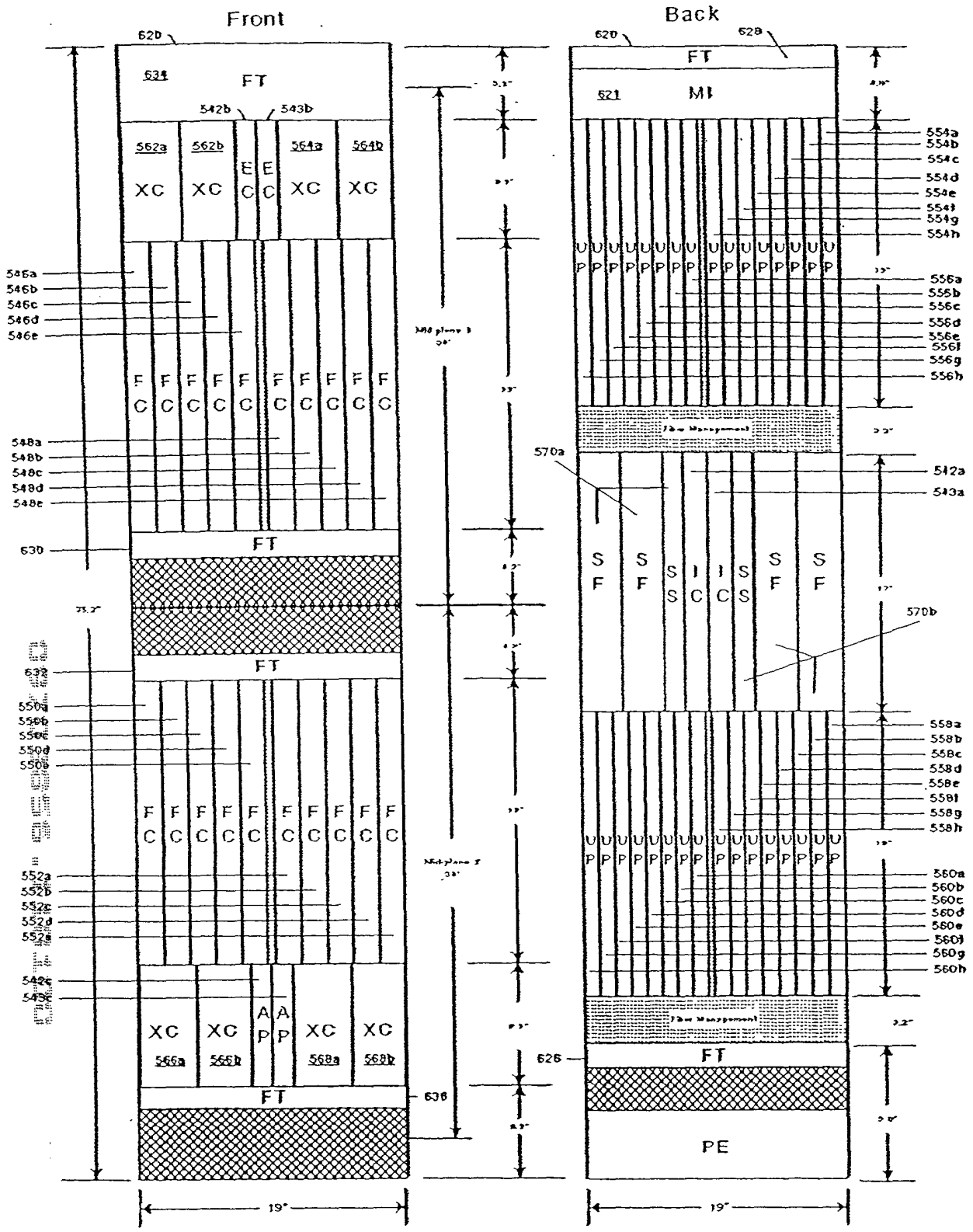


Fig. 41A

Fig. 41B

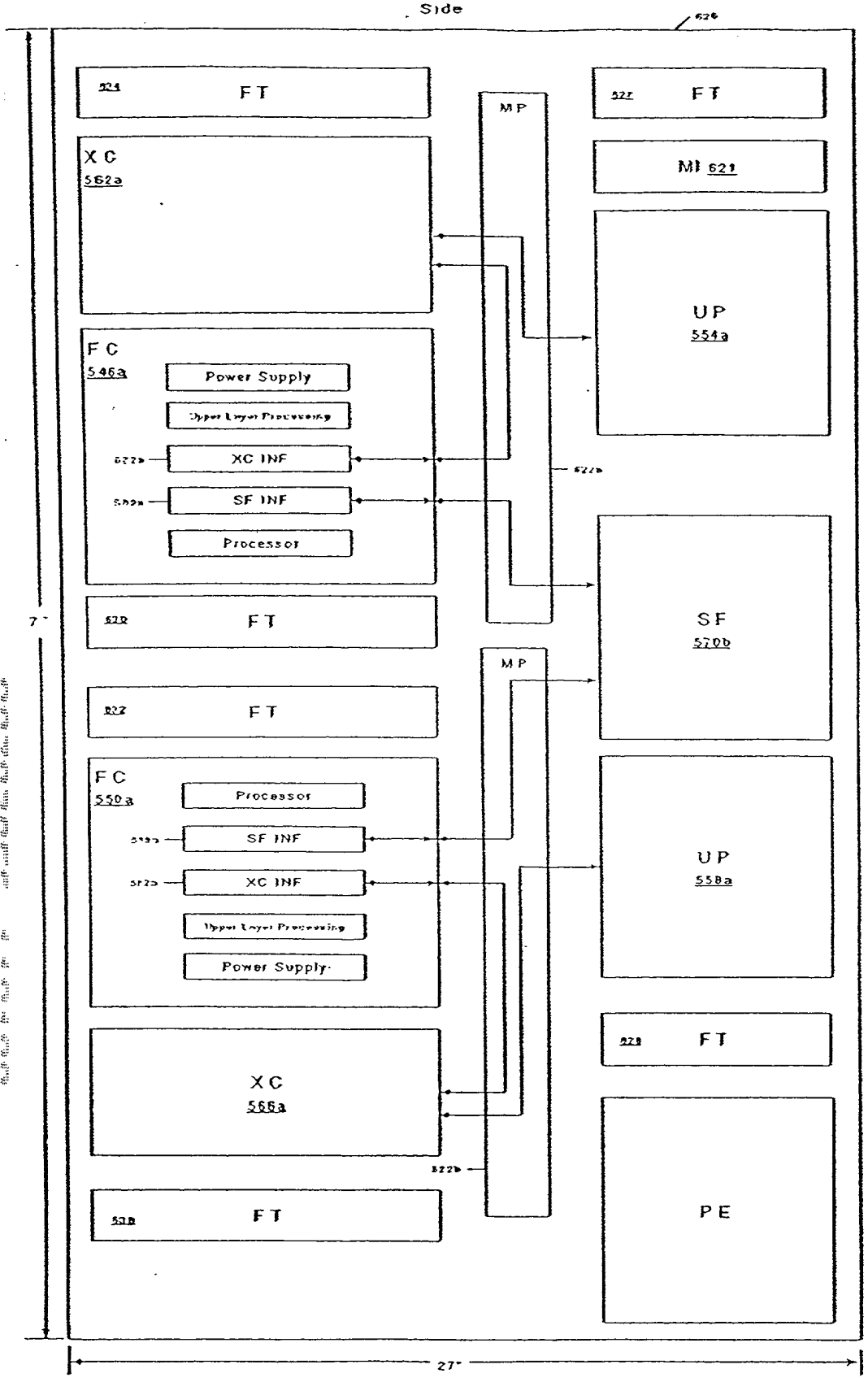

Side
626
524
FT
522
FT
MP
MI 621
XC
562a
UP
554a
FC
546a
Power Supply
Upper Layer Processing
522a
XC INF
522b
568a
SF INF
Processor
520
FT
SF
570b
MP
522
FT
FC
550a
Processor
568a
SF INF
522a
XC INF
Upper Layer Processing
Power Supply
UP
558a
528
FT
XC
566a
522b
PE
518
FT
7"
27"

Fig. 41C

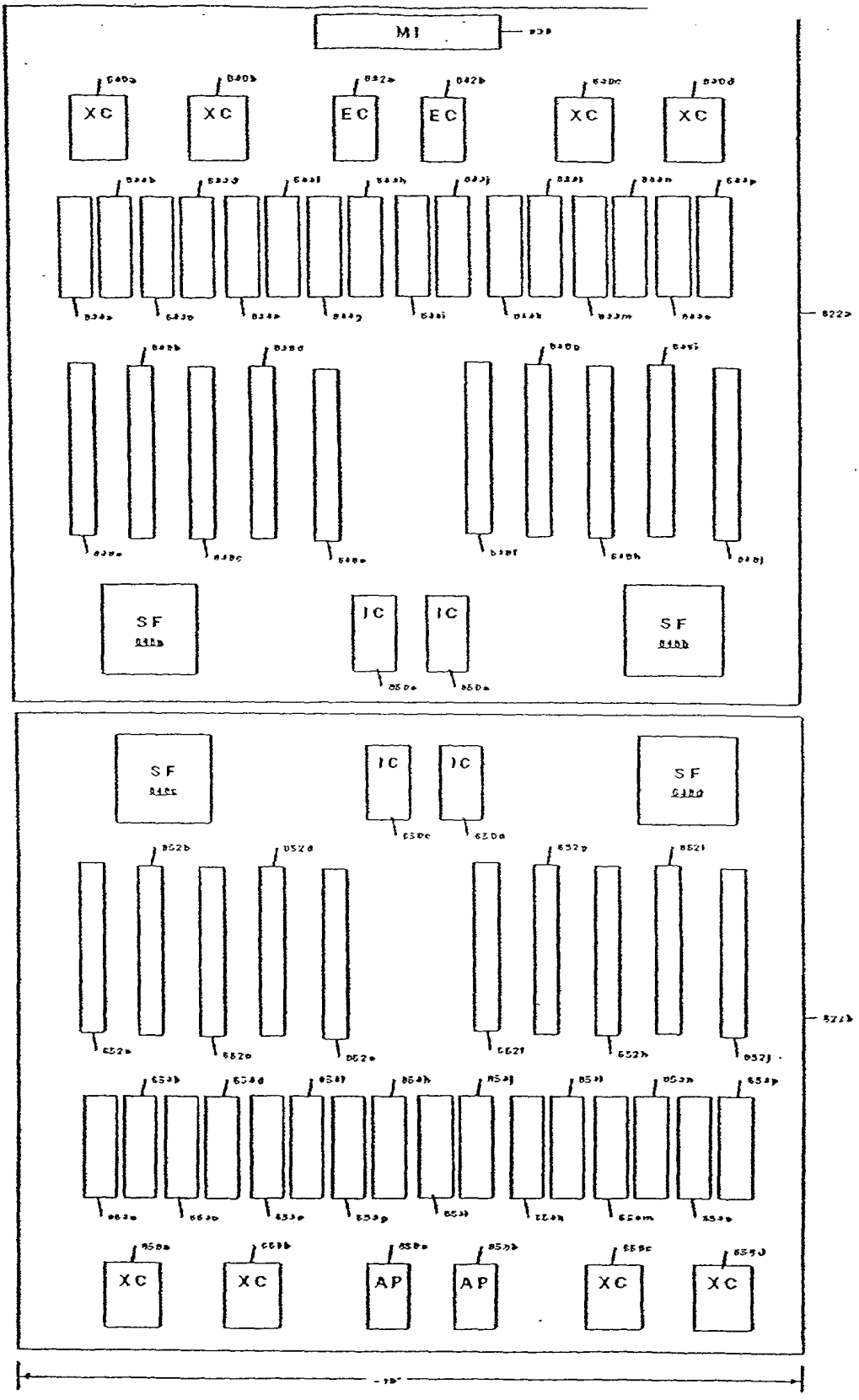
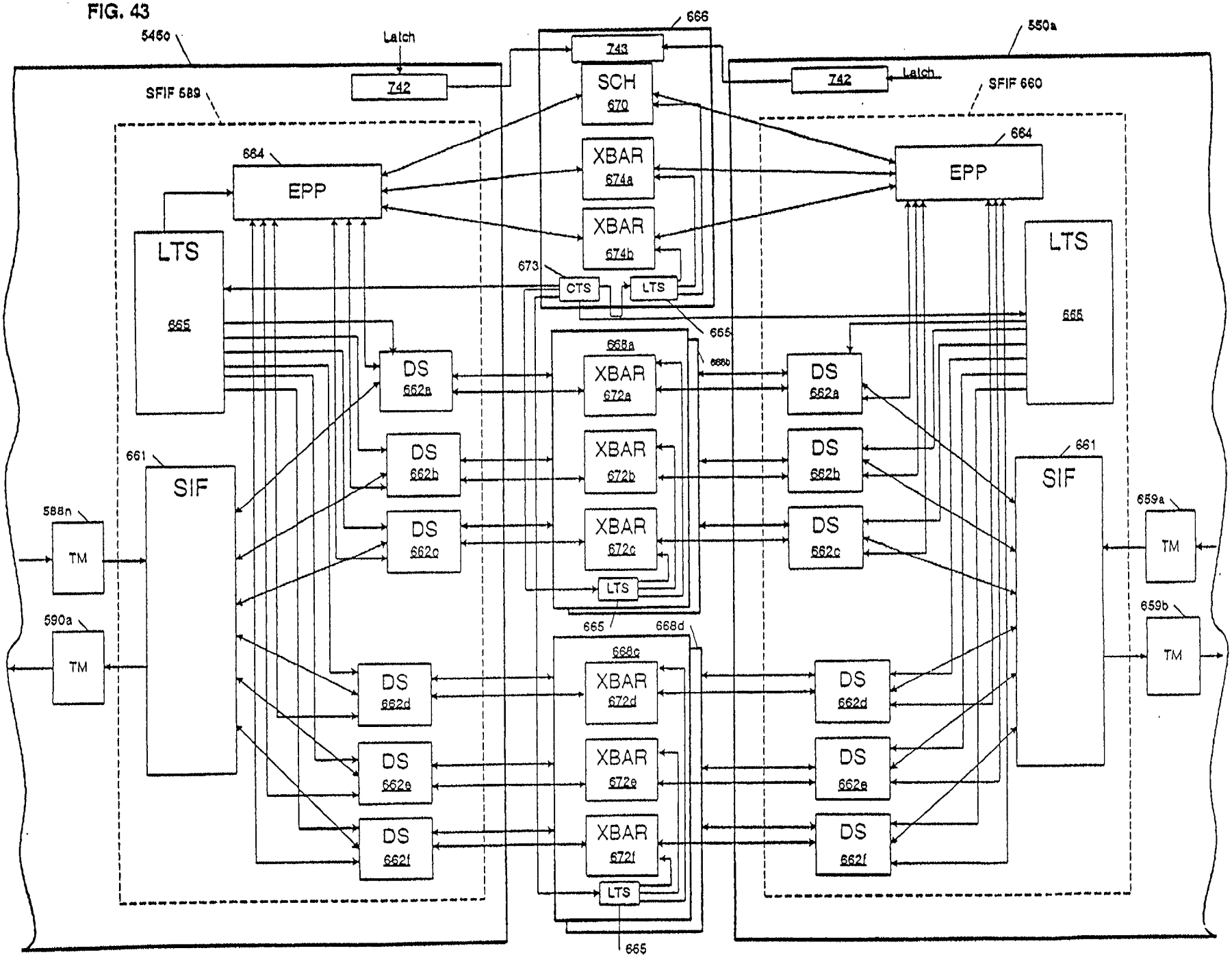


Fig. 42

PATENT PENDING

FIG. 43



Sent By: Equipe Communications; 781 795 2031; Jul-6-00 4:25PM; Page 3

DRAFT - 09/29/2000

Sent By: Equipe Communications; 781 795 2031; Jul-6-00 4:26PM; Page 4

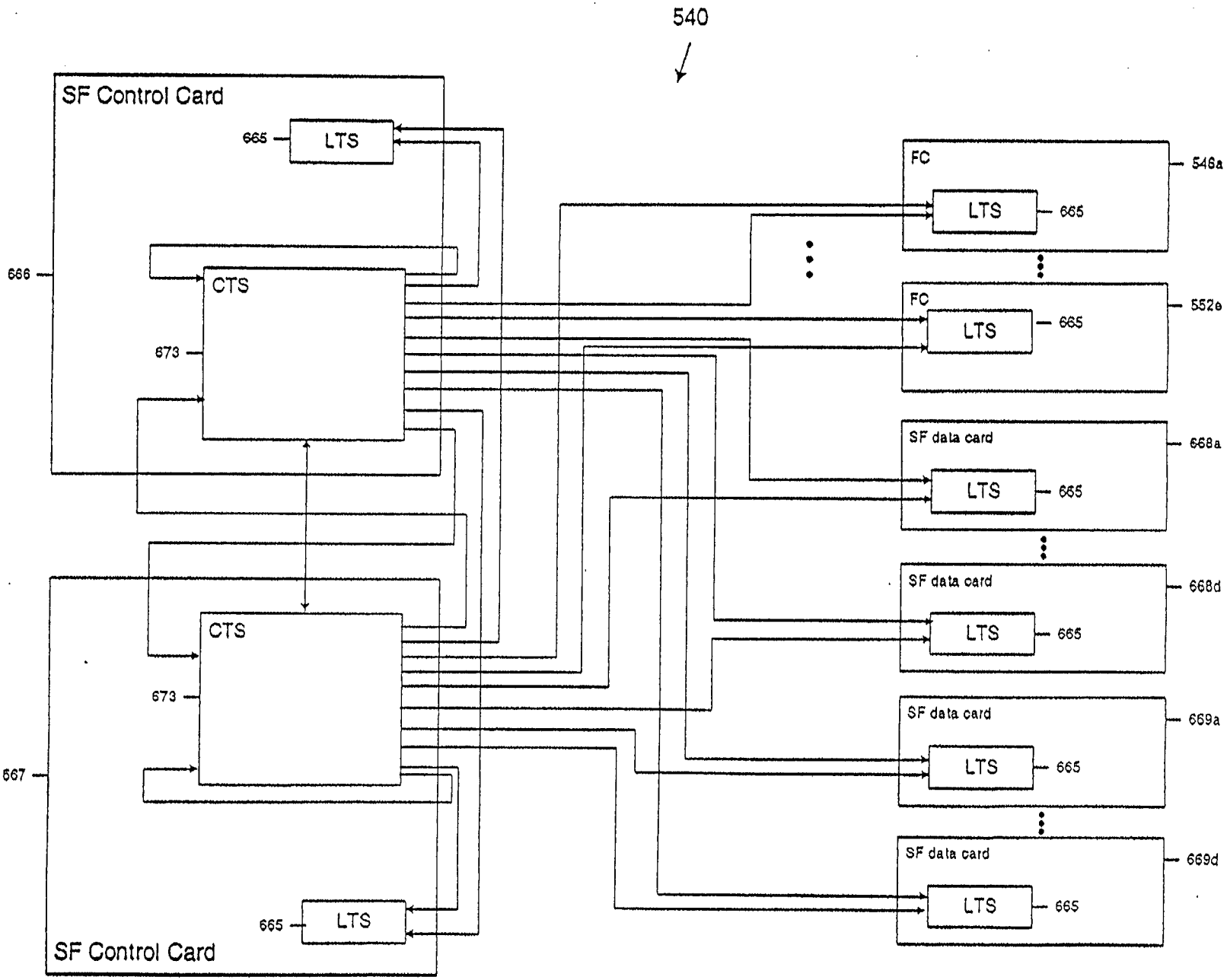
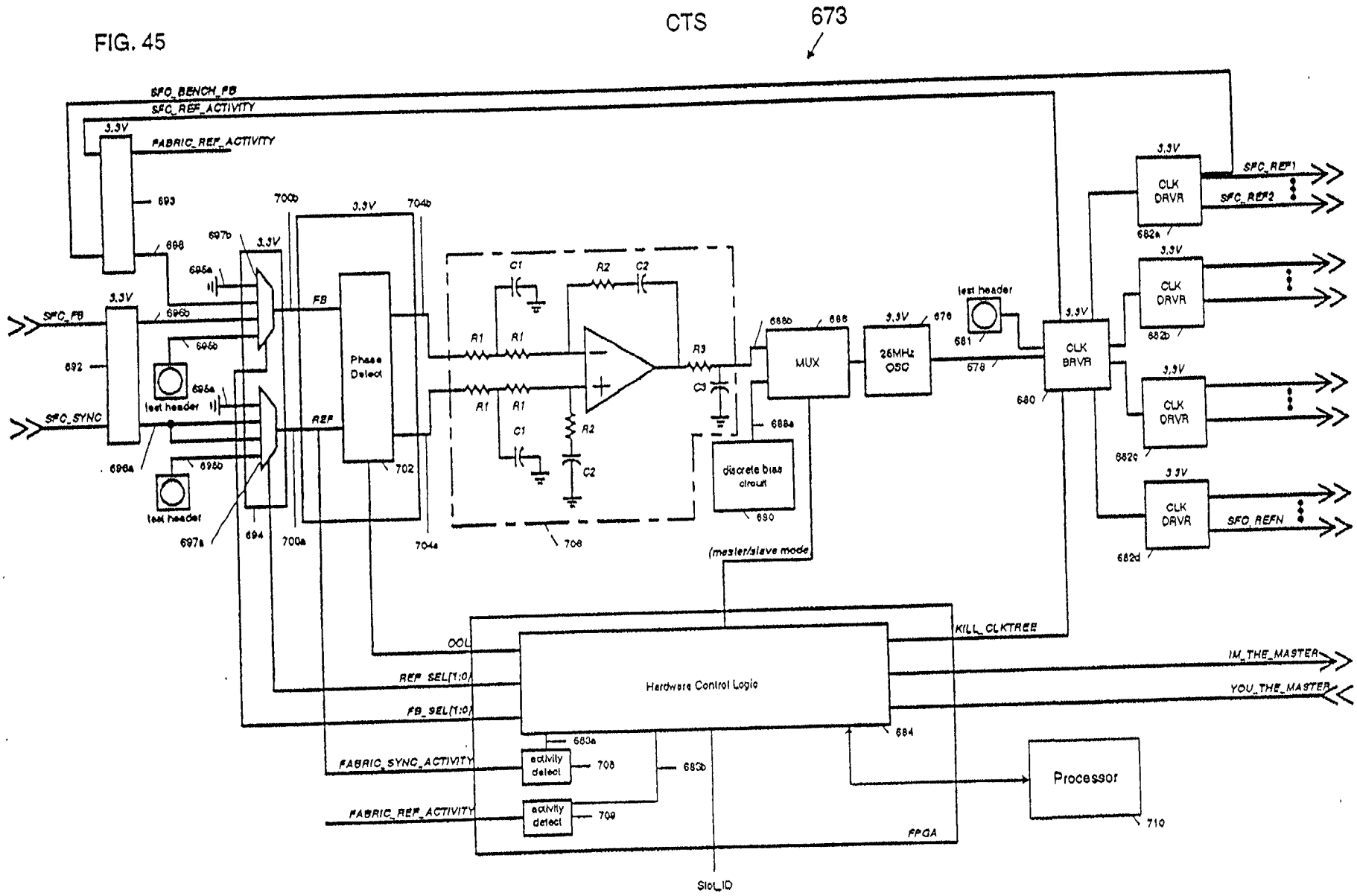


FIG. 44

PATENT DESIGNED

FIG. 45



Sent By: Equipe Communications; 781 795 2031; Jul-6-00 4:26PM; Page 5

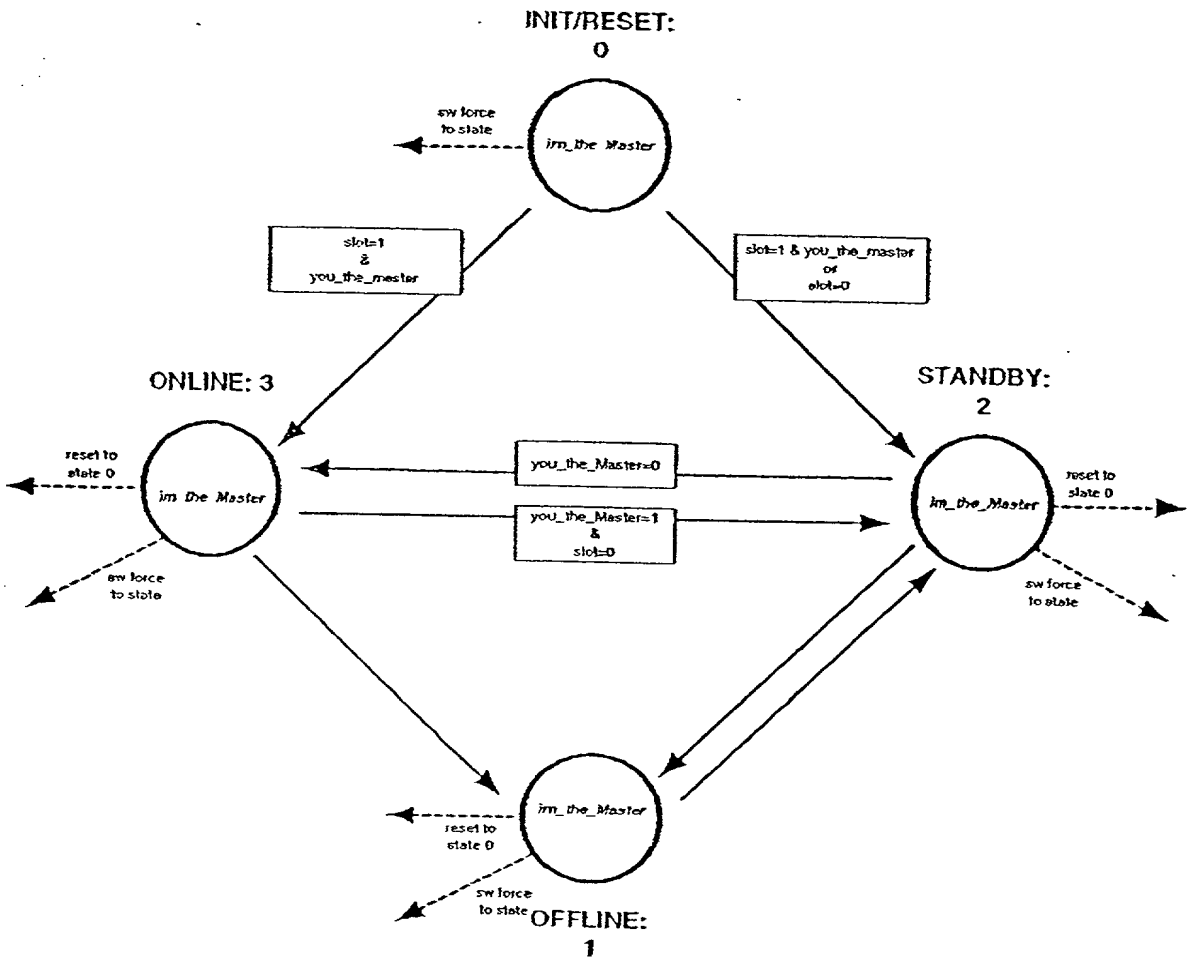

INIT/RESET:
0
sw force
to state
im_the_Master
slot=1
&
you_the_master
slot=1 & you_the_master
or
slot=0
ONLINE: 3
STANDBY:
2
reset to
state 0
im_the_Master
you_the_Master=0
you_the_Master=1
&
slot=0
im_the_Master
reset to
state 0
sw force
to state
im_the_Master
reset to
state 0
sw force
to state
im_the_Master
reset to
state 0
sw force
to state
OFFLINE:
1

FIG. 46

DRAFT - DESIGNED

Sent By: Equipe Communications; 781 795 2031; Jul-6-00 4:26PM; Page 7

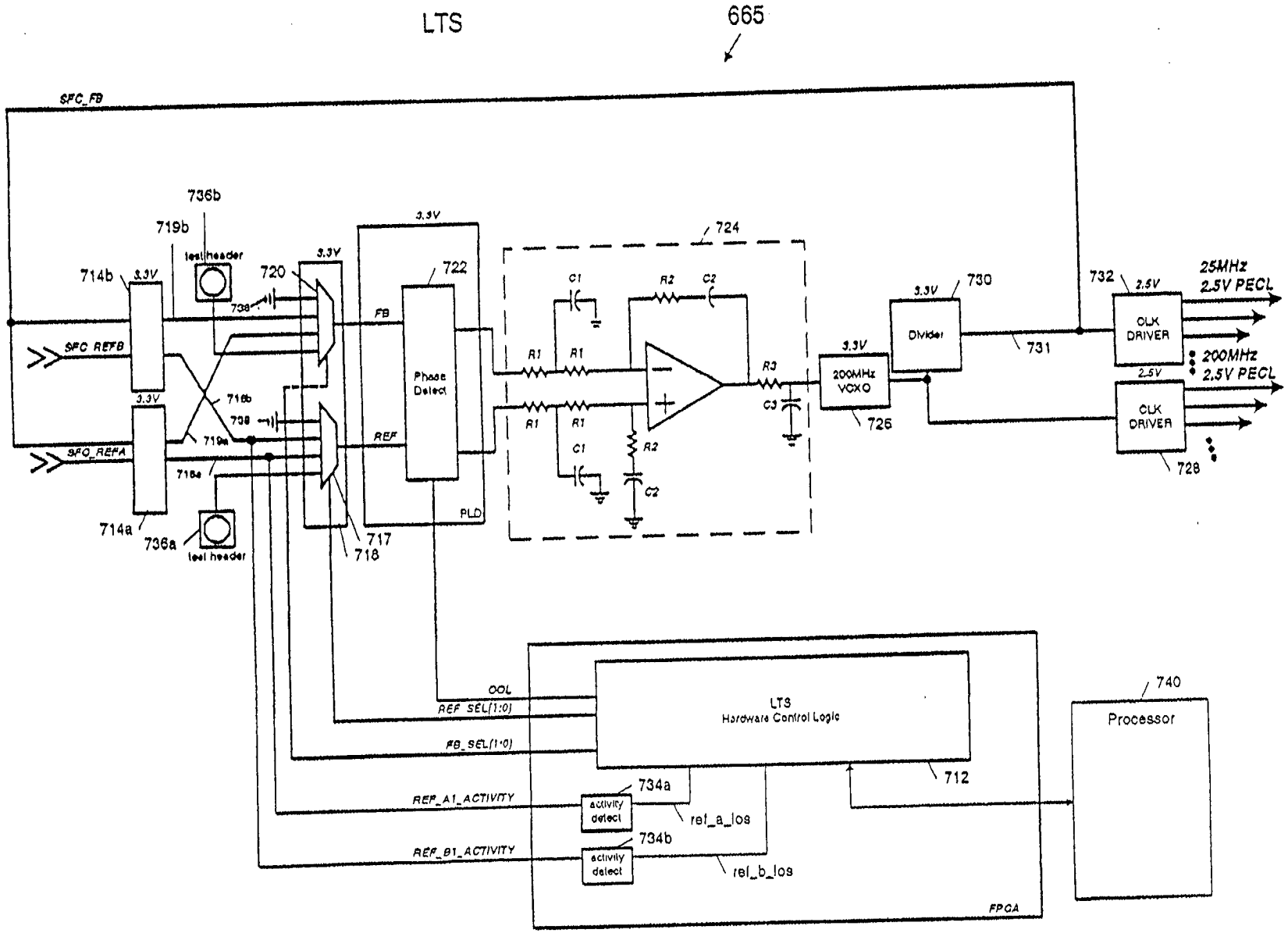


FIG. 47

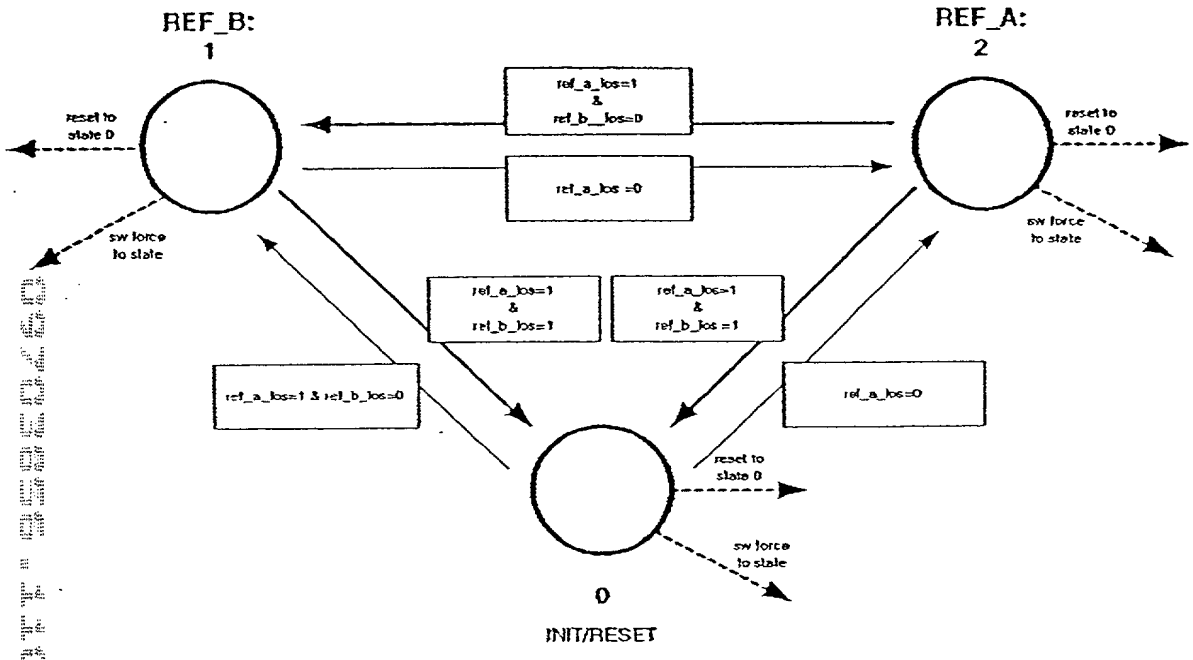

REF_B:
1
REF_A:
2
reset to state 0
sw force to state
ref_a_los=1 & ref_b_los=0
ref_a_los =0
ref_a_los=1 & ref_b_los=1
ref_a_los=1 & ref_b_los =1
ref_a_los=1 & ref_b_los=0
ref_a_los=0
reset to state 0
sw force to state
0
INIT/RESET

FIG. 48

Sent By: Equipe Communications; 781 795 2031; Jul-6-00 4:27PM; Page 9

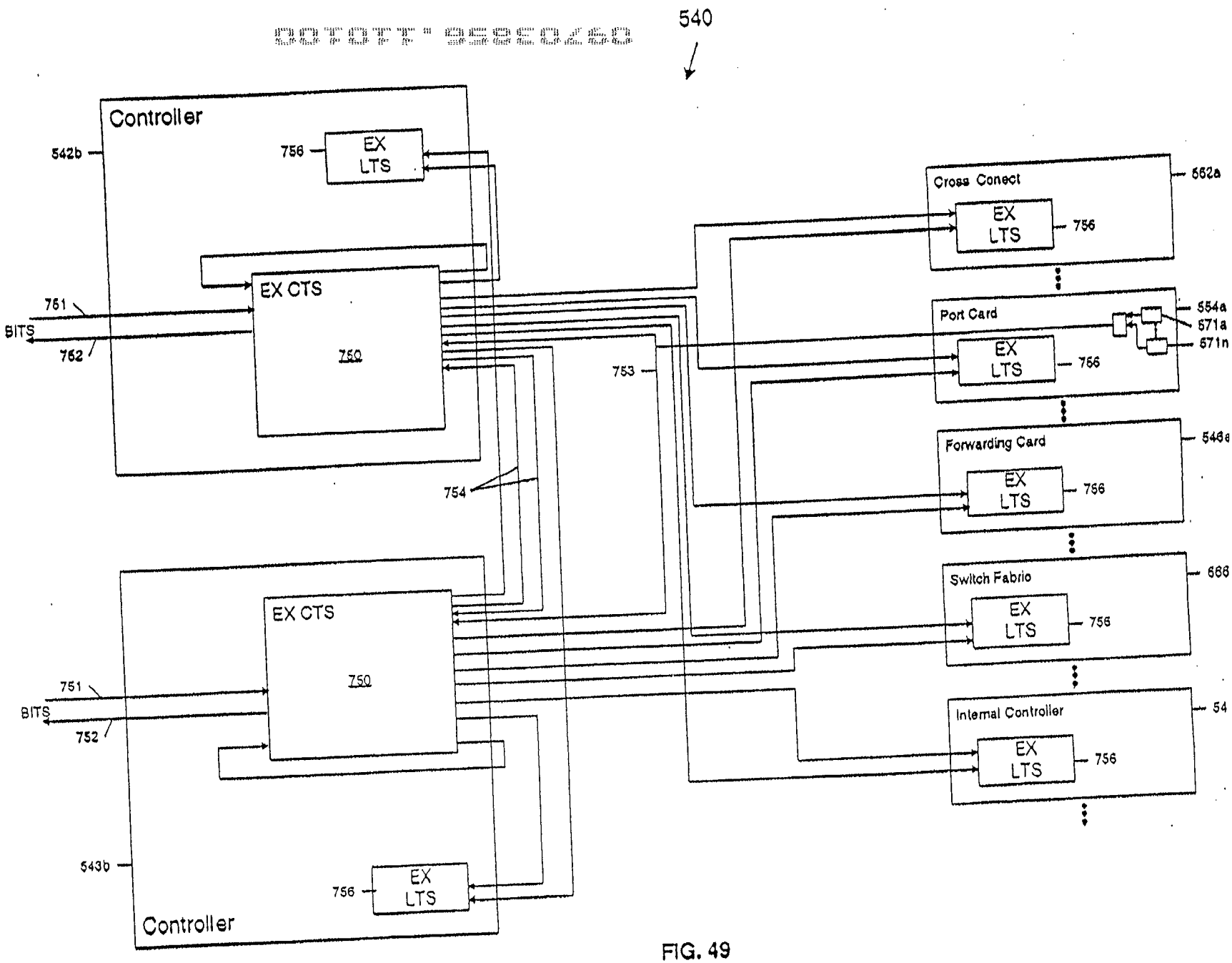


FIG. 49

Sent By: Equipe Communications; 781 795 2031; Jul-6-00 4:27PM; Page 10

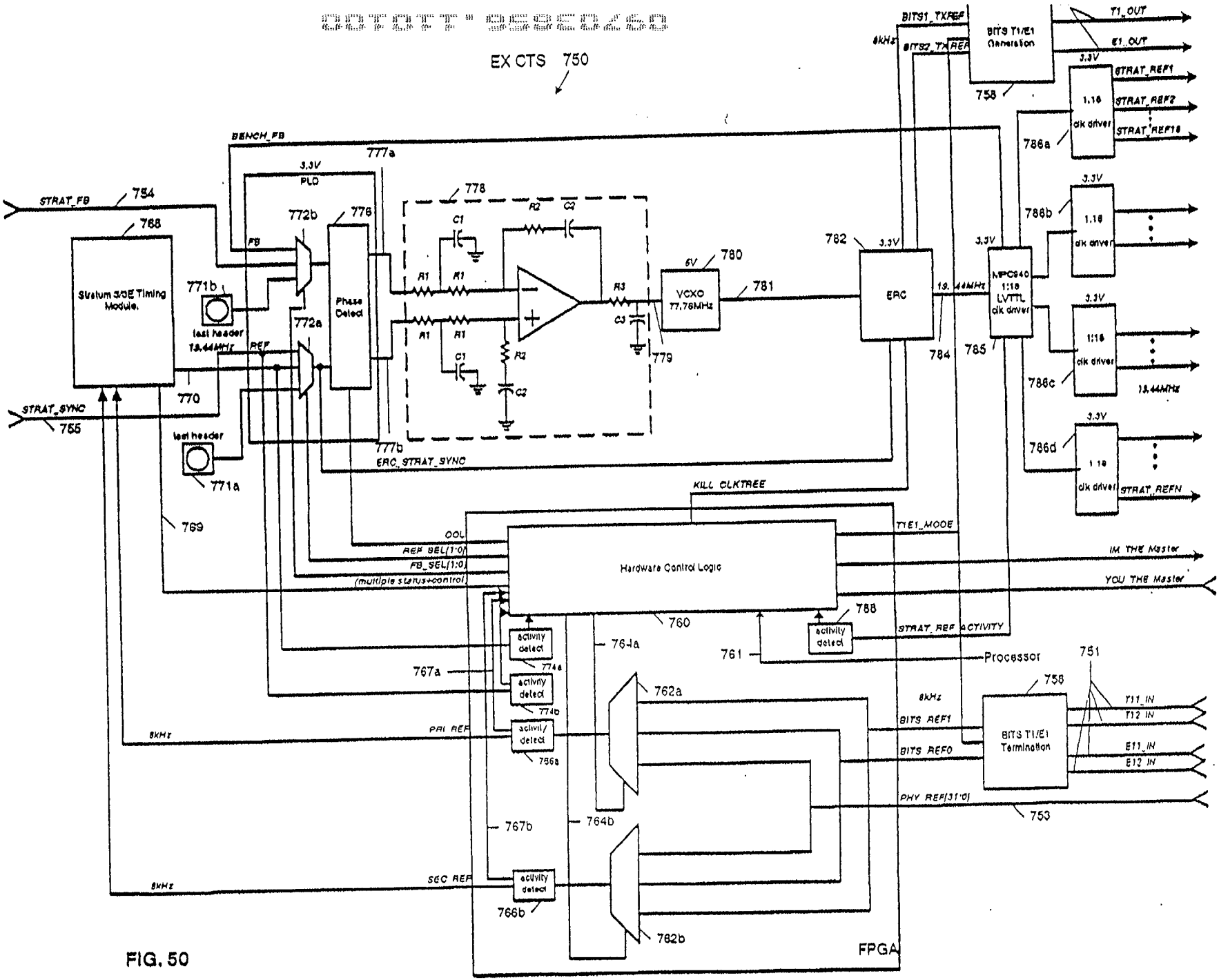


FIG. 50

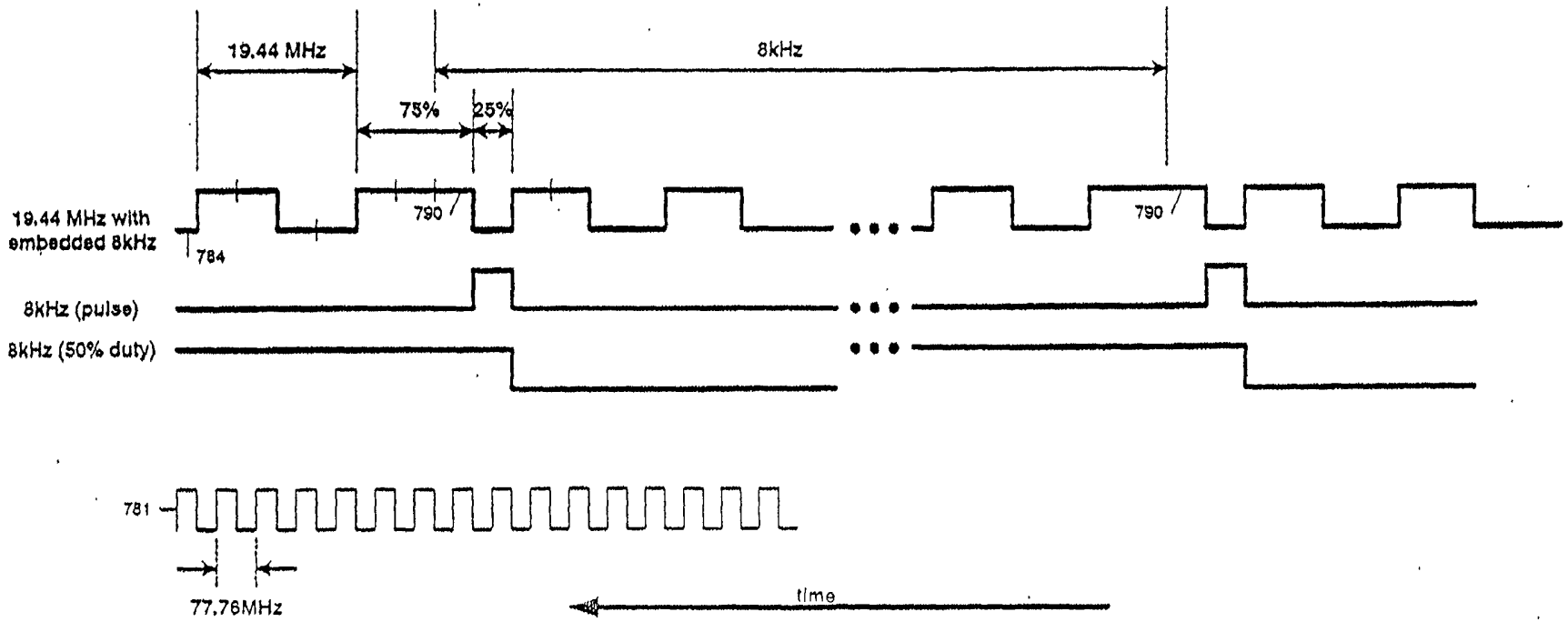

19.44 MHz
8kHz
75%
25%
19.44 MHz with
embedded 8kHz
784
790
790
8kHz (pulse)
8kHz (50% duty)
781
77.76MHz
time

FIG. 51

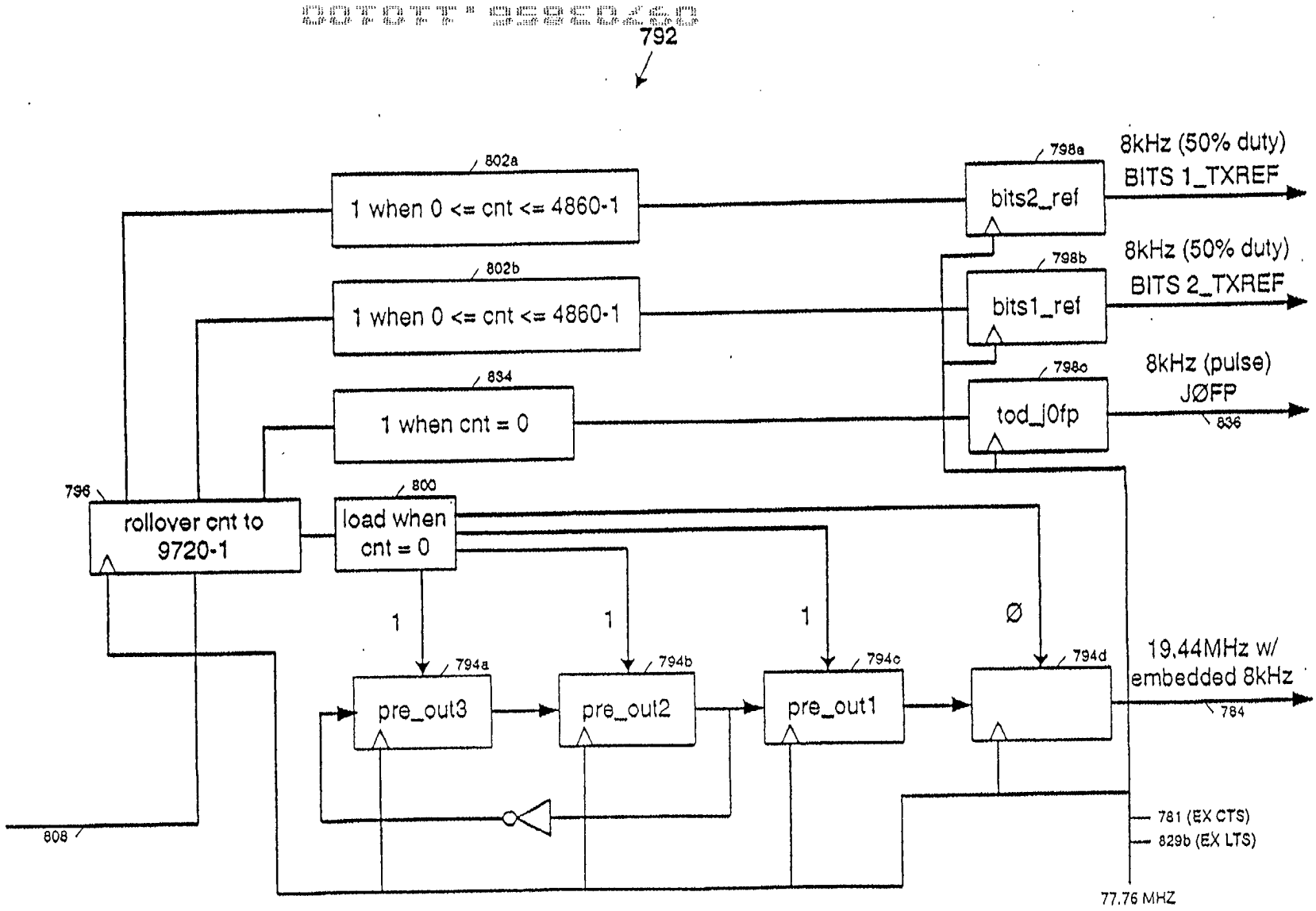
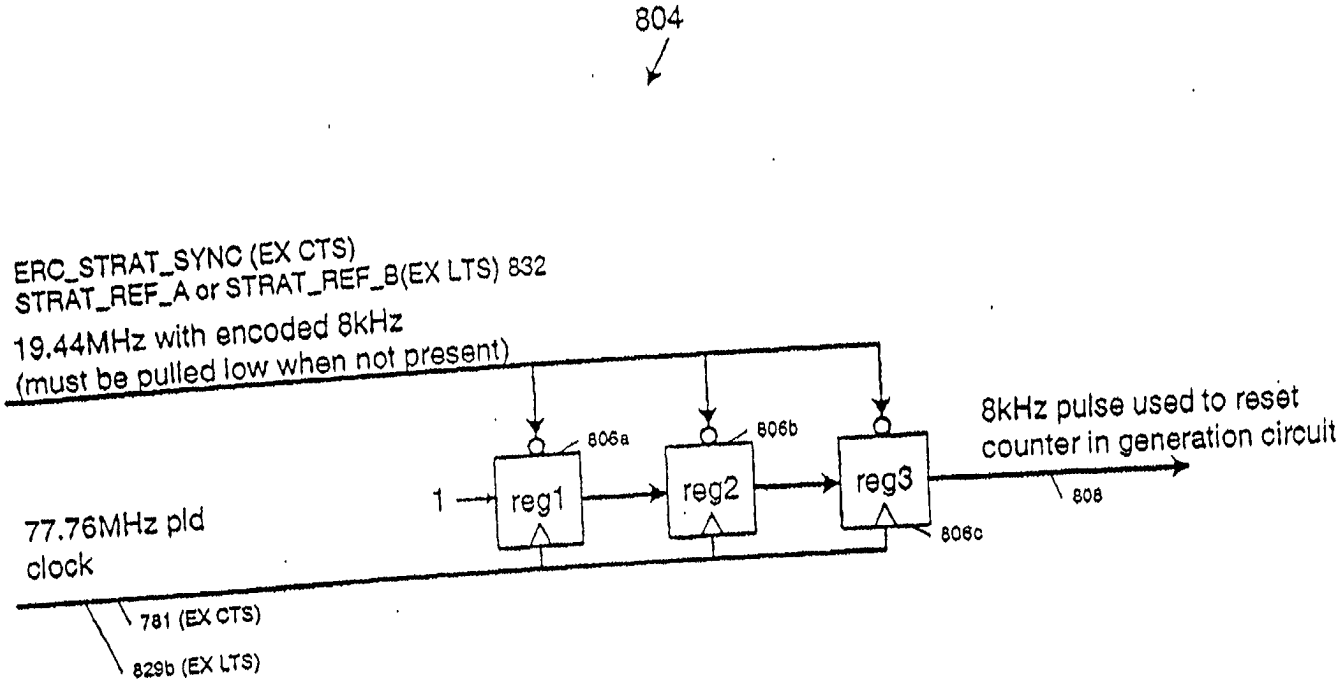

792
802a
1 when 0 <= cnt <= 4860-1
802b
1 when 0 <= cnt <= 4860-1
834
1 when cnt = 0
796
rollover cnt to 9720-1
800
load when cnt = 0
798a
bits2_ref
8kHz (50% duty)
BITS 1_TXREF
798b
bits1_ref
8kHz (50% duty)
BITS 2_TXREF
798c
tod_j0fp
8kHz (pulse)
JØFP
836
1
1
1
Ø
794a
pre_out3
794b
pre_out2
794c
pre_out1
794d
19.44MHz w/ embedded 8kHz
784
808
781 (EX CTS)
829b (EX LTS)
77.76 MHZ

Fig. 52

Sent By: Equipe Communications; 781 795 2031; Jul-6-00 4:28PM; Page 13/15



Extractor

FIG. 53

Sent By: Equipe Communications; 781 795 2031; Jul-6-00 4:28PM; Page 14/15

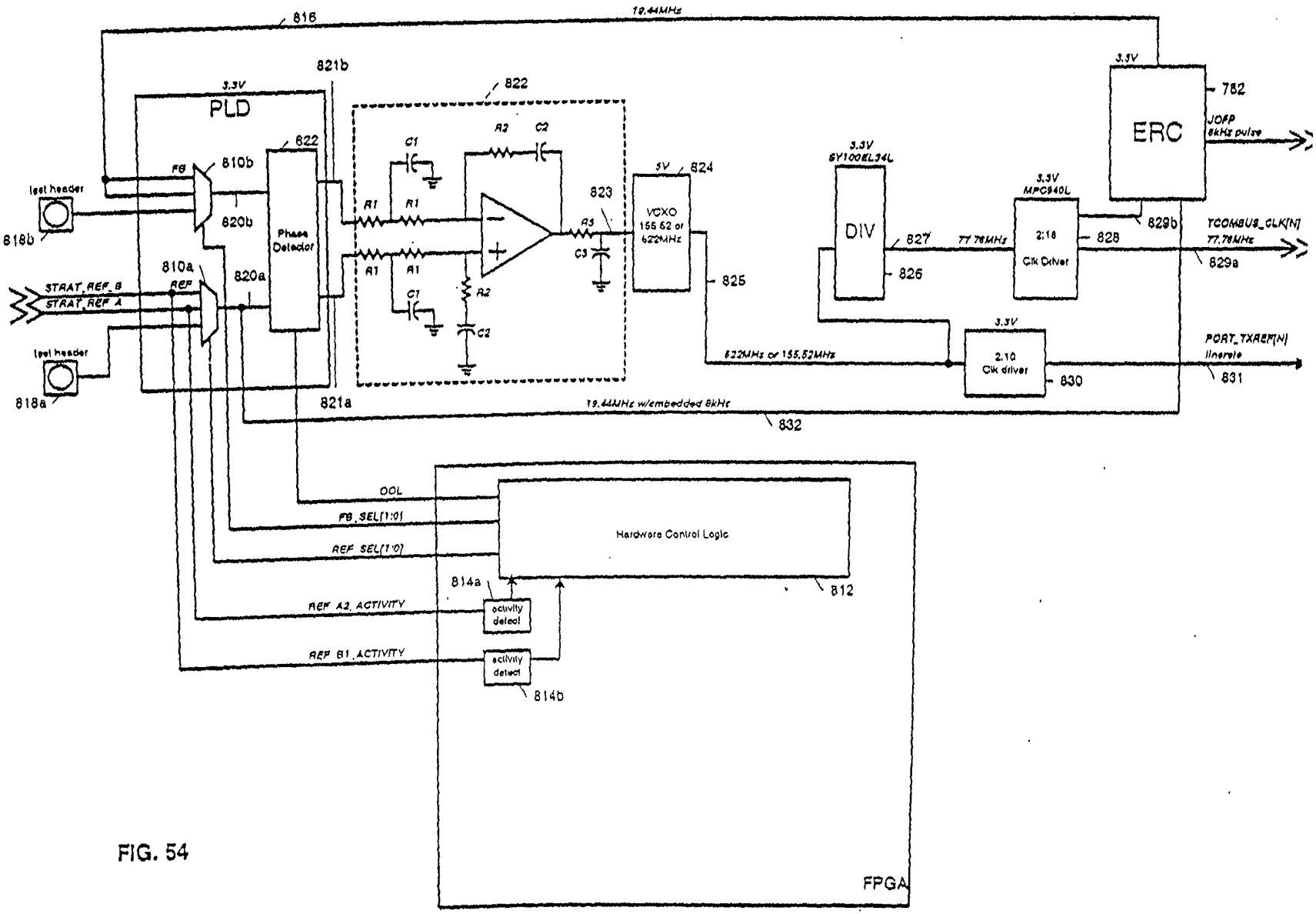


FIG. 54

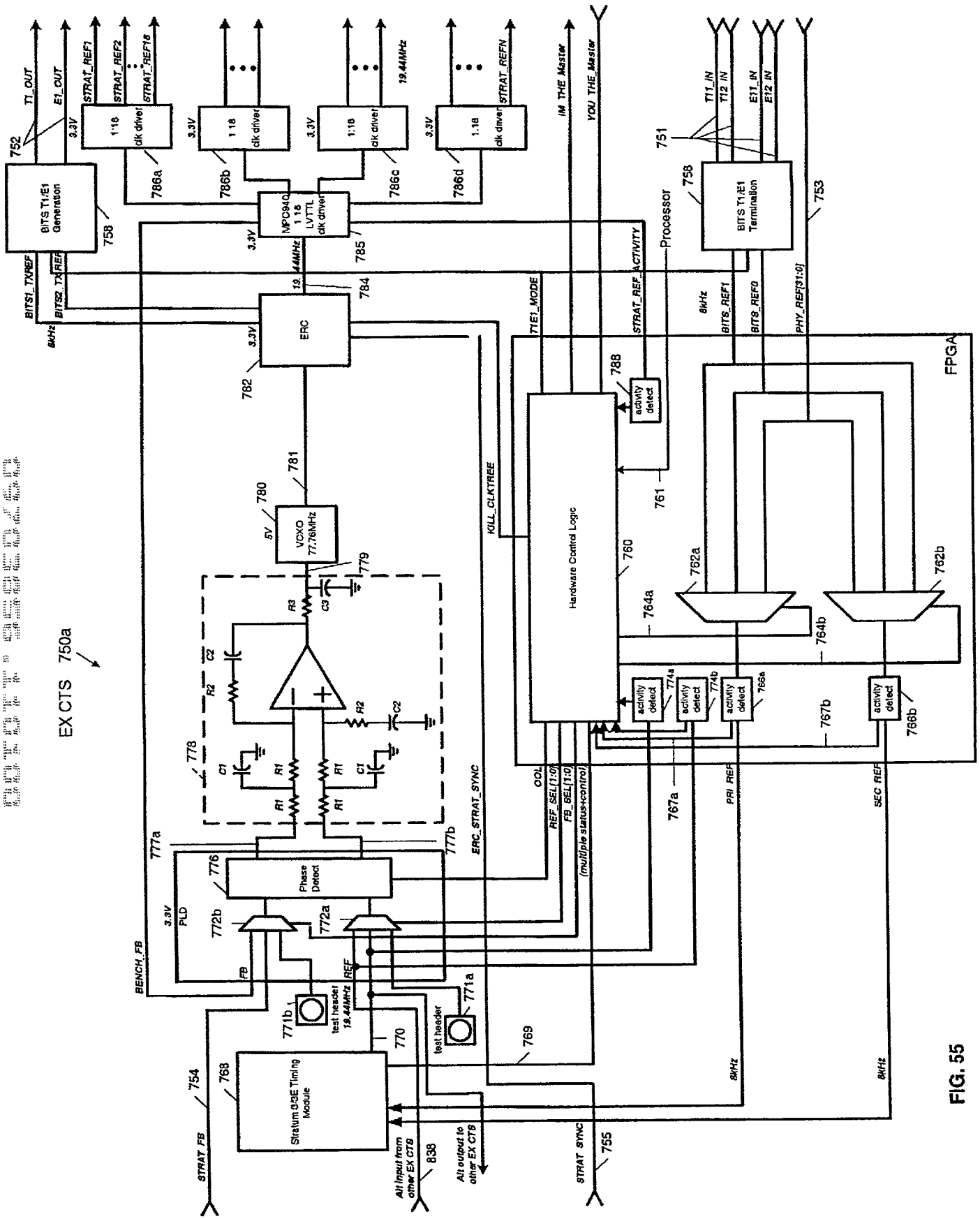

EX CTS 750a
T1_OUT
E1_OUT
STRAT_REF1
STRAT_REF2
STRAT_REF16
STRAT_REFN
19.44MHz
IM THE Master
YOU THE Master
T11_IN
T12 IN
E11_IN
E12 IN
BITS T1/E1 Generation
BITS T1/E1 Termination
MPC940 1:18 LVTTL clk driver
clk driver
1:18
3.3V
5V
752
758
753
751
786a
786b
786c
786d
785
784
782
781
780
779
778
788
761
760
762a
762b
764a
764b
774a
774b
766a
766b
767a
767b
BITS1_TXREF
BITS2_TXREF
8kHz
ERC
VCXO 77.76MHz
Processor
Hardware Control Logic
activity detect
T1E1_MODE
STRAT_REF_ACTIVITY
BITS_REF1
BITS_REF0
PHY_REF[31:0]
FPGA
KILL_CLKTREE
ERC_STRAT_SYNC
REF_SEL[1:0]
FB_SEL[1:0]
(multiple status+control)
PRI REF
SEC REF
C1
C2
C3
R1
R2
R3
777a
777b
776
Phase Detect
PLD
772a
772b
BENCH_FB
FB
REF
771a
771b
test header
test header 19.44MHz
770
769
754
768
Stratum 3/3E Timing Module
STRAT FB
STRAT SYNC
755
All input from other EX CTS
All output to other EX CTS
838

FIG. 55

DRAFT: 05/05/2000

Sent By: Equipe Communications; 781 795 2031; Jul-12-00 3:57PM; Page 3/5

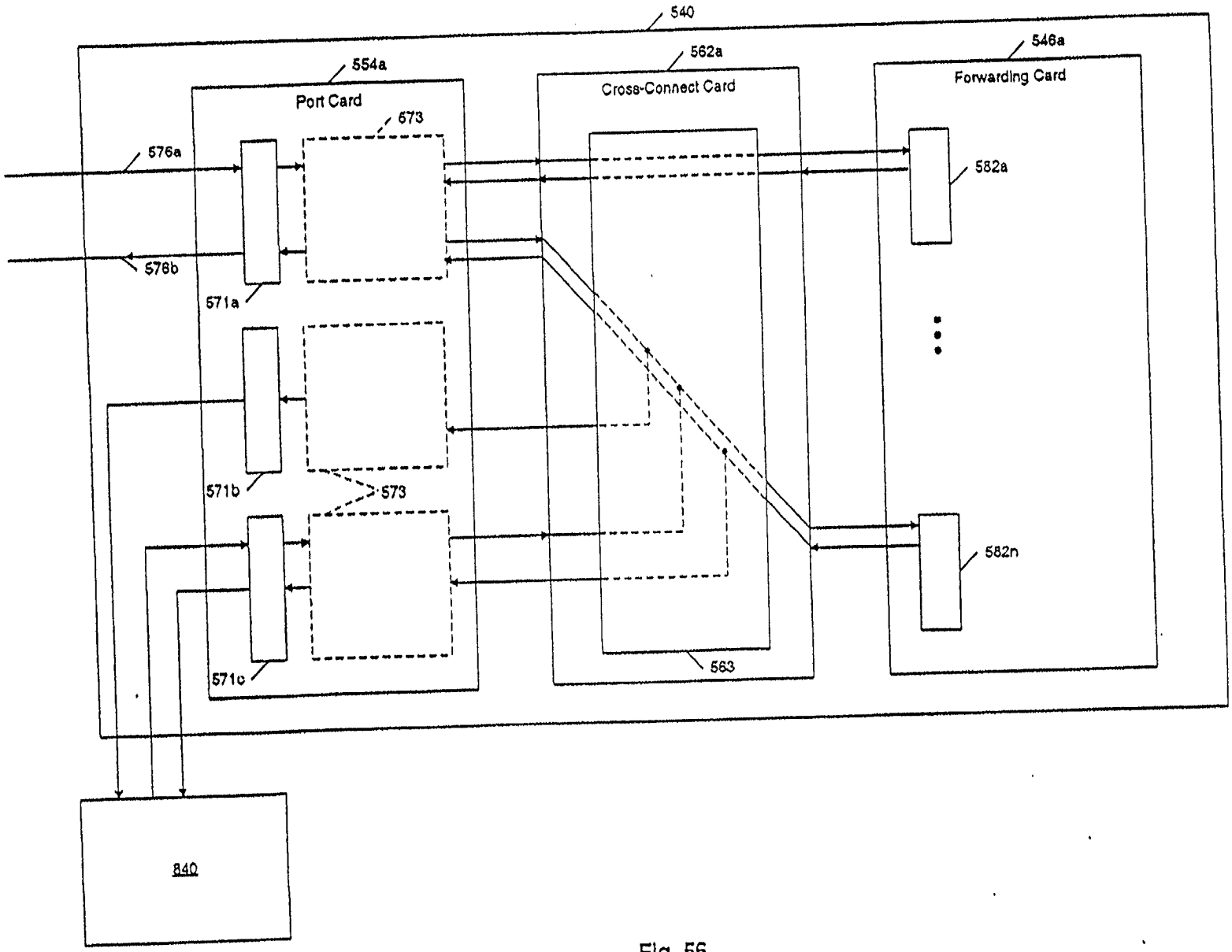
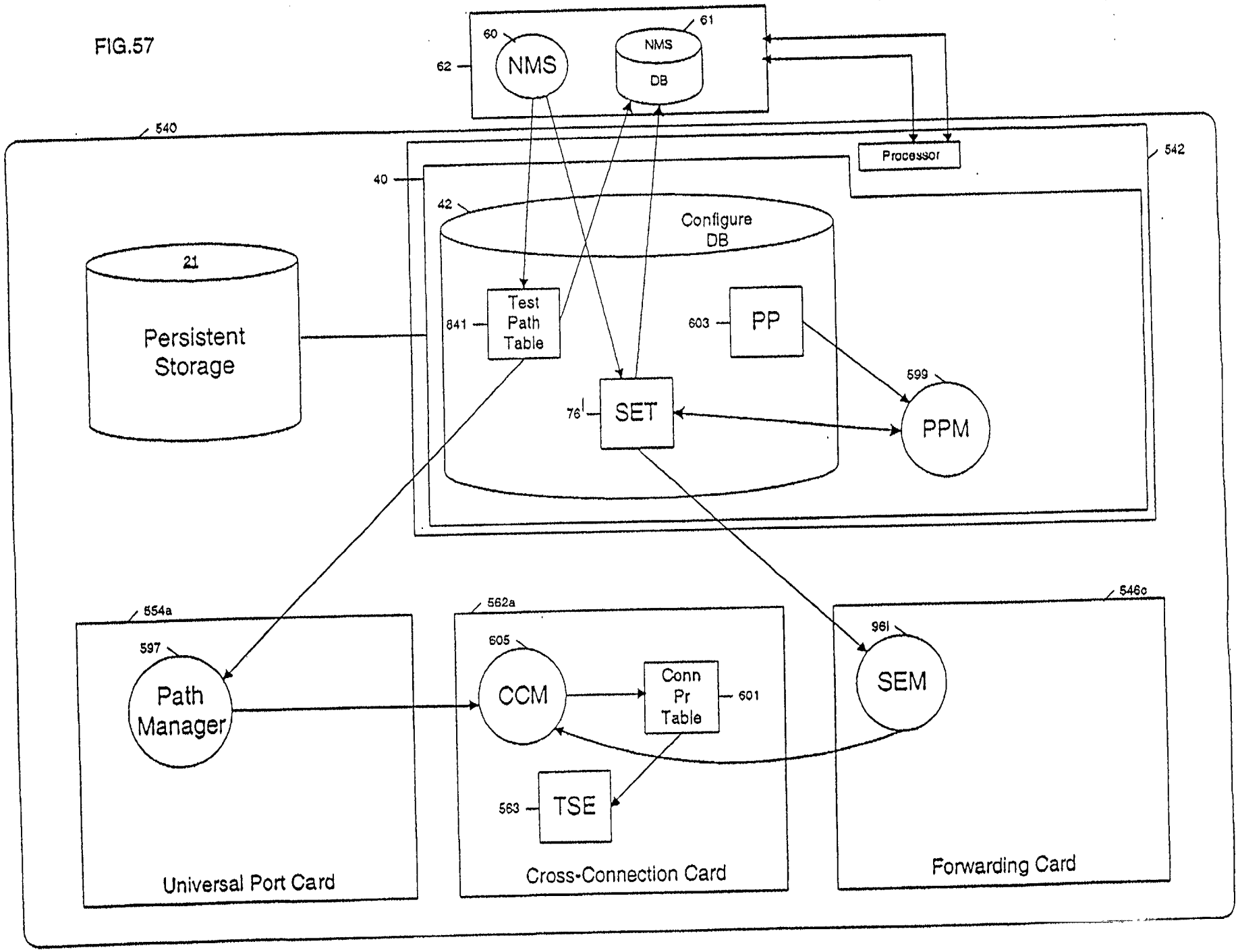


Fig. 56

Sent By: Equipe Communications; 781 795 2031; Jul-12-00 3:57PM; Page 4/5

FIG.57



DRAFT DELETED

Sent By: Equipe Communications; 781 795 2031; Jul-12-00 3:57PM; Page 5/5

FIG. 58

Test Path Table 841

	Path LID	UP Port LID	Time Slot	# of Time Slots	Monitor (844)	Enable Port Receiver (845)	...
842	1666	1232	4	3	Ingress	No	
843	1666	1233	4	3	Egress	No	
844	1666	1233	4	3	Ingress	Yes	
	⋮	⋮	⋮	⋮			⋮

DRAFT - 09/29/2000

Sent By: Equipe Communications; 781 795 2031; Oct-9-00 6:54PM; Page 5

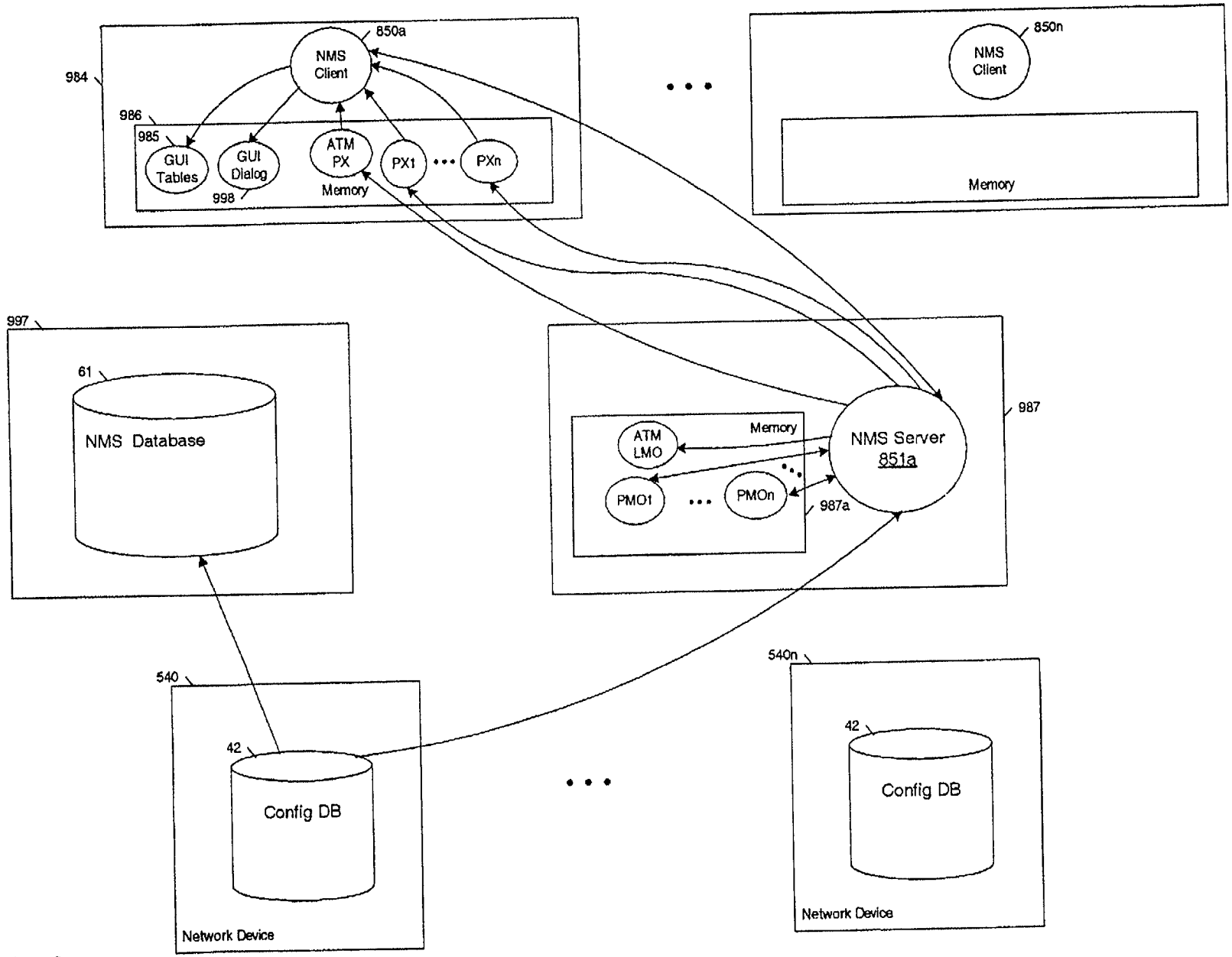


Fig. 59

Fig. 60a

Managed Device Table

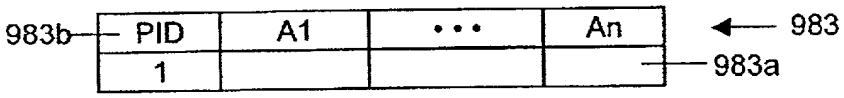


Fig. 60b

Chassis Table

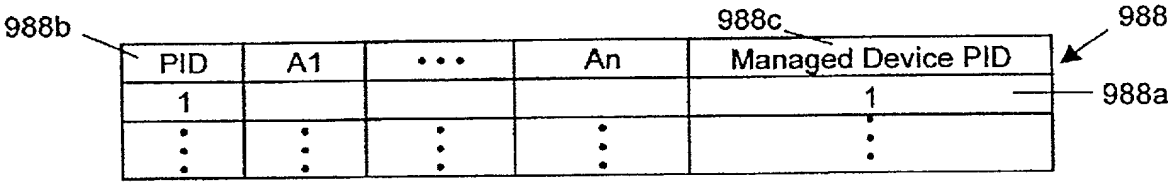


Fig. 60c

Shelf Table

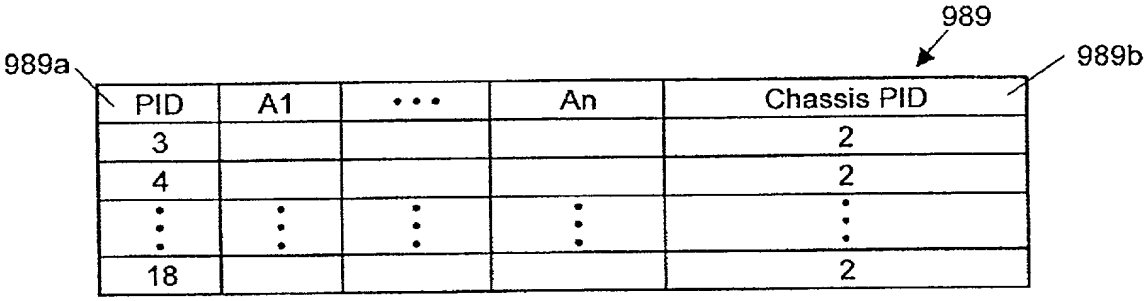
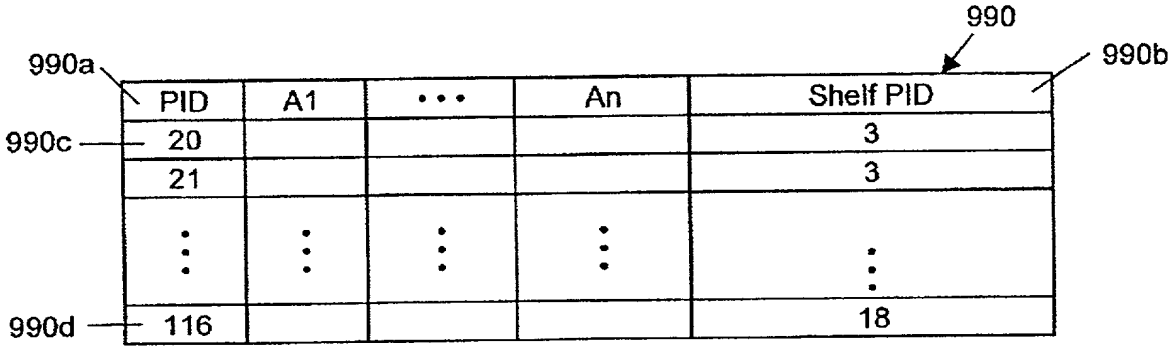


Fig. 60d

Slot Table



Sent By: Equipe Communications; 781 795 2031; Oct-9-00 6:54PM; Page 6

Sent By: Equipe Communications; 781 795 2031; Oct-9-00 6:54PM; Page 7/16

DRAFT DELETED

CARD TABLE

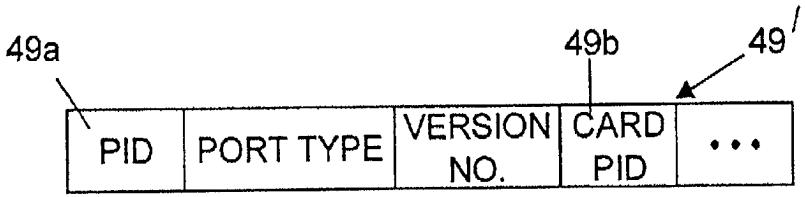
47a 47b 47'

PID	CWD TYPE	VERSION NO.	SLOT PID	•••
120	0XF002	3	20	
121	0XF002	4	21	
⋮	⋮	⋮	⋮	⋮
124	0X6002	1	24	
⋮	⋮	⋮	⋮	⋮
131	0XF002	1	31	
⋮	⋮	⋮	⋮	⋮

FIG. 60e

Sent By: Equipe Communications; 781 795 2031; Oct-9-00 6:55PM; Page 8/16

PORT TABLE



PID	PORT TYPE	VERSION NO.	CARD PID	...
300	00620	1	20	
301	00620	1	20	
302	00620	1	20	
303	00620	1	20	
304	00820	1	20	
⋮	⋮	⋮	⋮	⋮
400	OO620	1	39	
⋮	⋮	⋮	⋮	⋮

FIG. 60f

PATENT SEARCH

Sent By: Equipe Communications; 781 795 2031; Oct-9-00 6:55PM; Page 9/16

FIG. 60g

SONET Path Table 600

600a 600b

Path LID	Port PID	Time Slot	# of Time Slots	•••
901	304	4	3	
⋮	⋮	⋮	⋮	⋮

Sent By: Equipe Communications; 781 795 2031; Oct-9-00 6:55PM; Page 10/16

FIG. 60h

Service EndPoint Table 76 //

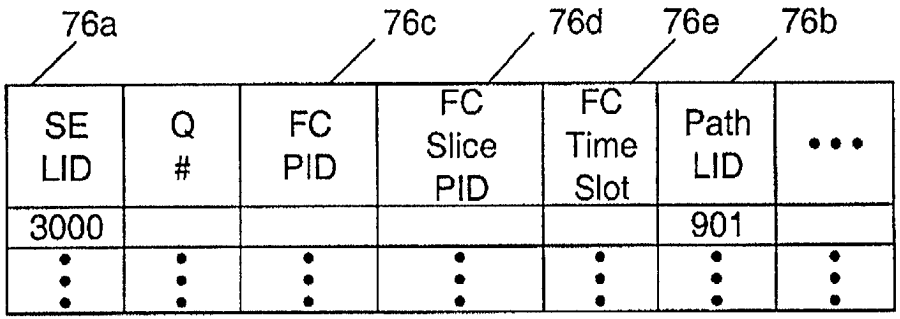
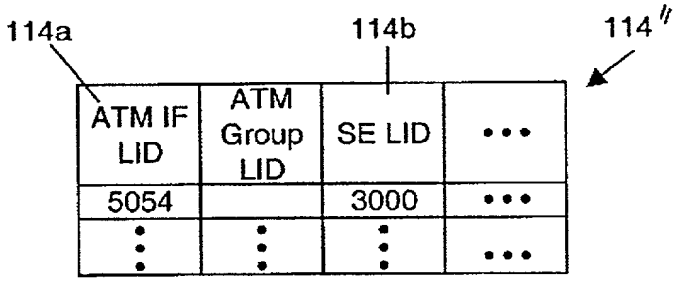


Fig. 60i

ATM IF Table



Sent By: Equipe Communications; 781 795 2031; Oct-9-00 6:55PM; Page 11/16

Fig. 60j

Virtual ATM IF Table

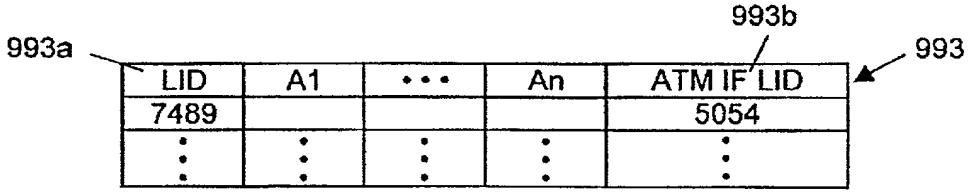


Fig. 60k

Virtual Connection Table

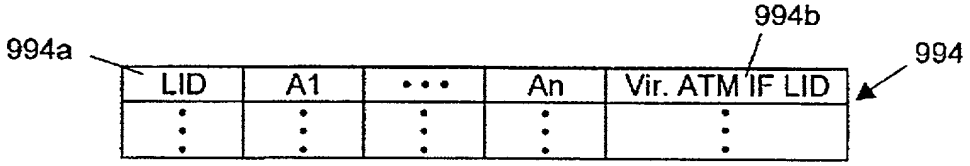


Fig. 60L

Virtual Link Table

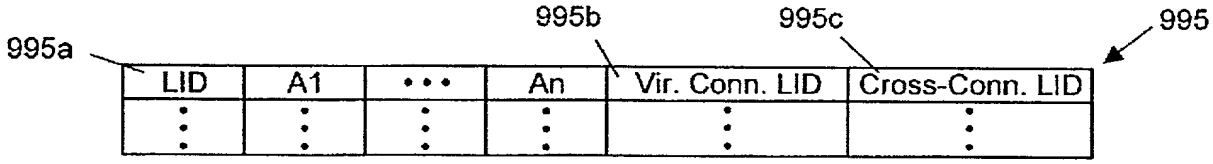
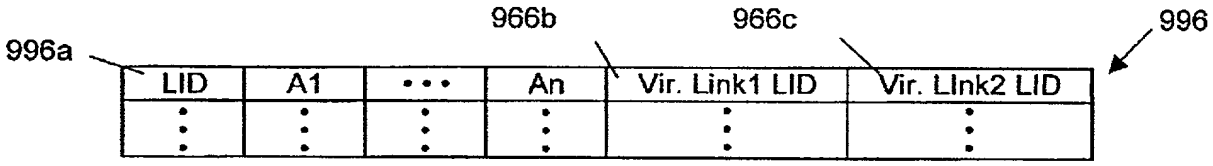


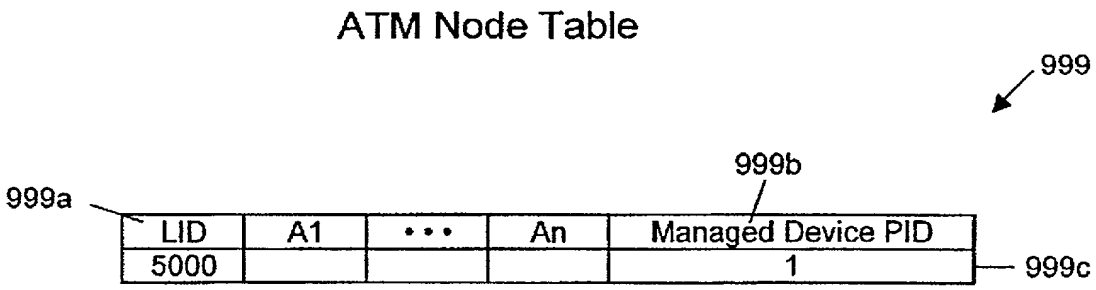
Fig. 60m

Cross-Connect Table



Sent By: Equipe Communications; 781 795 2031; Oct-9-00 6:55PM; Page 12/16

Fig. 60n

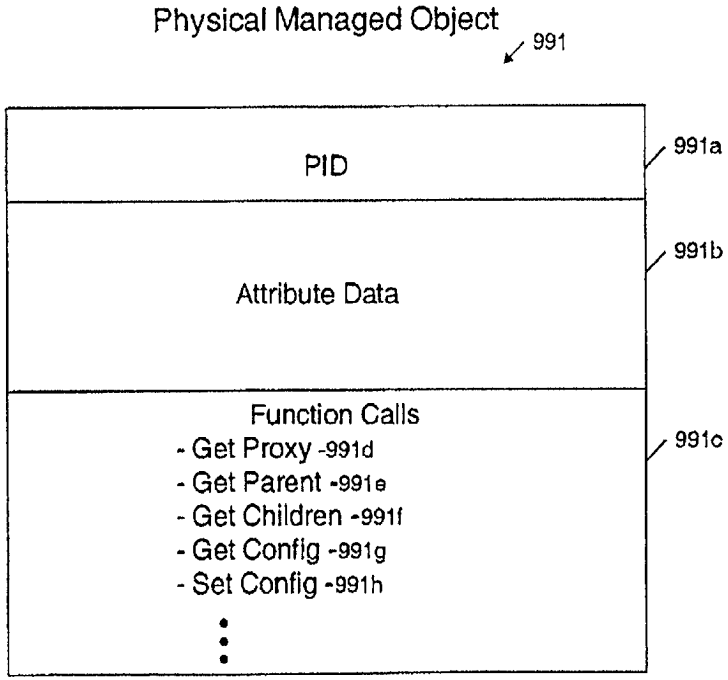


Sent By: Equipe Communications; 781 795 2031; Oct-9-00 6:56PM; Page 13/16

PATENT DESIGN

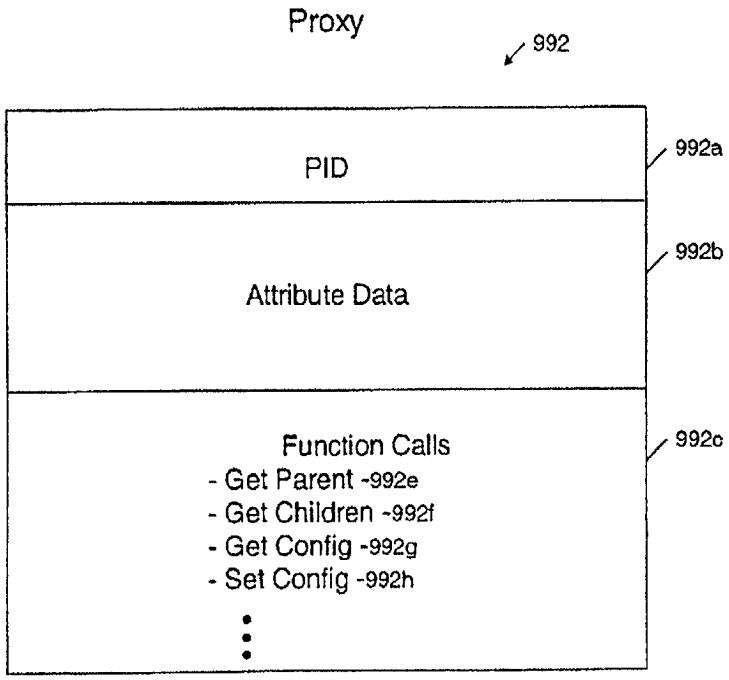
Sent By: Equipe Communications; 781 795 2031; Oct-9-00 6:56PM; Page 14/16

Fig. 61a



Sent By: Equipe Communications; 781 795 2031; Oct-9-00 6:56PM; Page 15/16

Fig. 61b



997

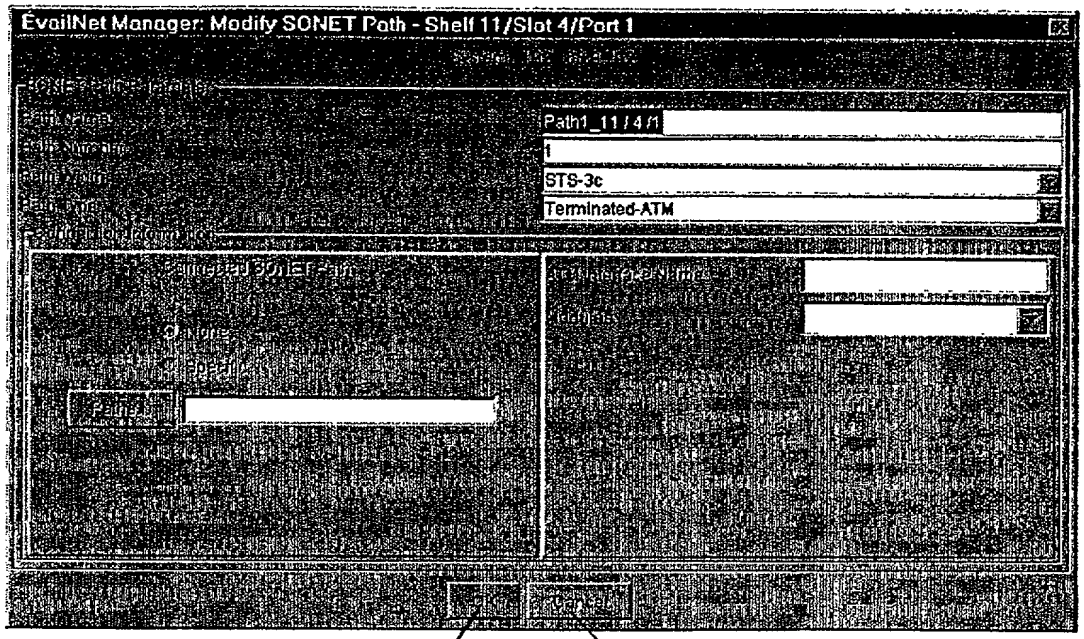

EvailNet Manager: Modify SONET Path - Shelf 11/Slot 4/Port 1
Path1_11/4/1
1
STS-3c
Terminated-ATM
Terminated SONET Path
None
Path1
Path1
998b
997a

Fig. 62

EXPRESS MAILING LABEL NO.: EL327514237US

Attorney Docket No.: 102689-61/00-U0082

DECLARATION AND POWER OF ATTORNEY FOR UNITED STATES LETTERS PATENT APPLICATION

As a below-named inventor, I hereby declare that:

My residence, post-office address and citizenship are as stated below next to my name.

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

ACCESSING NETWORK DEVICE DATA THROUGH USER PROFILES

the specification of which

(check one)

☒ is attached hereto.

☐ was filed on: Herewith

as Application No.: Pending

and was amended on:
(if applicable).

In the event that the filing date and/or Application No. are not entered above at the time I execute this document, and if such information is deemed necessary, I hereby authorize and request my attorneys/agent(s) at Nutter, McClennen & Fish, LLP, One International Place, Boston, MA 02110-2699, to insert above the filing date and/or Application No. of said application.

I hereby state that I have reviewed and understand the contents of the above-identified application specification, including the claims, as amended by any amendment specifically referred to herein.

I acknowledge the duty to disclose all information known to me that is material to patentability in accordance with Title 37, Code of Federal Regulations, §1.56.

Atty. Dkt. No. 102689-61/00-U0082

FOREIGN PRIORITY CLAIM

I hereby claim foreign priority benefits under Title 35, United States Code §119(a)-(d) of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

(check one)

☒ no such foreign applications have been filed.

☐ such foreign applications have been filed as follows:

EARLIEST FOREIGN APPLICATION(S), IF ANY FILED WITHIN 12 MONTHS (6 MONTHS FOR DESIGN) PRIOR TO THIS U.S. APPLICATION

Country	Application Number	Date of Filing (month, day, year)	Priority Claimed Under 35 USC 119
			___ Yes No ___
			___ Yes No ___
			___ Yes No ___
			___ Yes No ___
			___ Yes No ___

ALL FOREIGN APPLICATION(S), IF ANY FILED MORE THAN 12 MONTHS (6 MONTHS FOR DESIGN) PRIOR TO THIS U.S. APPLICATION

Atty. Dkt. No. 102689-61/00-U0082

CLAIM FOR BENEFIT OF EARLIER U.S. PROVISIONAL APPLICATION(s)

I hereby claim priority benefits under Title 35, United States Code §119(e), of any United States provisional patent application(s) listed below:

(check one)

☒ no such U.S. provisional applications have been filed.

☐ such U.S. provisional applications have been filed as follows:

Application Number	Date of Filing (month, day, year)	Priority Claimed Under 35 USC 119(e)
		___ Yes No ___
		___ Yes No ___
		___ Yes No ___

CLAIM FOR BENEFIT OF EARLIER U.S./PCT APPLICATION(s)

I hereby claim the benefit under Title 35, United States Code §120, of the United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United State Code, §112, I acknowledge the duty to disclose all information that is material to patentability in accordance with Title 37, Code of Federal Regulations, §1.56, and which became available to me between the filing date of the prior application and the national or PCT international filing date of this application:

(check one)

☐ no such U.S./PCT applications have been filed.

☒ such U.S./PCT applications have been filed as follows:

Application Number	Date of Filing (month,day,year)	Status (Patented/Pending/Abandoned)
Pending	October 12, 2000	UTILIZING MANAGED OBJECT PROXIES IN NETWORK MANAGEMENT SYSTEMS

Atty. Dkt. No. 102689-61/00-U0082

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment or both under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

I hereby appoint:

Ronald E. Cahill	Reg. No. 38,403	Lisa J. Michaud	Reg. No. 44,238
Maria M. Eliseeva	Reg. No. 43,328	Reza Mollaaghababa	Reg. No. 43,810
Thomas J. Engellenner	Reg. No. 28,711	David J. Powsner	Reg. No. 31,868
Michael I. Falkoff	Reg. No. 30,833	Richard J. Roos	Reg. No. 45,053
William C. Geary III	Reg. No. 31,359	Scott D. Rothenberger	Reg. No. 41,277

all of Nutter, McClennen & Fish, LLP, One International Place, Boston, Massachusetts 02110-2699, jointly, and Patricia Davis, Reg. No. 37,866, of Équipe Communications Corporation, of 100 Nagog Park, Acton, MA 01720, and each of them severally, my attorneys at law/patent agent(s), with full power of substitution, delegation and revocation, to prosecute this application, to make alterations and amendments therein, to receive the patent, and to transact all business in the Patent and Trademark Office connected therewith.

Please mail correspondence to: Reza Mollaaghababa
at **Customer Number 021125**, whose address is:

Nutter McClennen & Fish, LLP
One International Place
Boston, Massachusetts 02110-2699

Please direct telephone calls to: Reza Mollaaghababa
at (617)439-2514.

Please direct facsimiles to: (617)310-9514

Atty. Dkt. No. 102689-61/00-U0082

Full name of sole or first joint inventor Darryl Black
Inventor's Signature D. P. Black Date 10-26-00
Residence 14 Hills Farm Lane, Hollis, NH 03049
Country of Citizenship U.S.
Post Office Address (required) same as above

Full name of second/joint inventor Thomas R. Perry
Inventor's Signature Thomas R. Perry Date 10-26-2000
Residence 230 Hayden Road, Groton, MA 01450
Country of Citizenship U.S.
Post Office Address (required) same as above

917811.1